January 2023

Quality Indicators for Administrative Data1

User Manual For R Package on GitHub

Sook Kim and Natalie Shlomo   
 University of Manchester

Version 1.2

1. Funded by the Grant: ‘Methodological Advancements on the Use of Administrative Data in Official Statistics’ ESRC (ES/V005456/1)

Contents

[1. Project aims and objectives 3](#_Toc124596017)

[2. Quality indicators 3](#_Toc124596018)

[2.1 Distance Metrics 4](#_Toc124596019)

[2.2 R-indicators 5](#_Toc124596020)

[3. User Guide on the R-package 7](#_Toc124596021)

[3.1 Download and inspect the contents 7](#_Toc124596022)

[3.1.1 Download 7](#_Toc124596023)

[3.1.2 Downloaded contents explained 8](#_Toc124596024)

[3.2 Launch RStudio and get ready 9](#_Toc124596025)

[3.2.1 Open the entire master folder in RStudio 9](#_Toc124596026)

[3.2.2 Set custom path 9](#_Toc124596027)

[3.2.3 Automatically create output folders 10](#_Toc124596028)

[3.2.4 Install packages 10](#_Toc124596029)

[3.3 RUNNING 2\_Prep\_Wtsample\_Freq\_Table.R 11](#_Toc124596030)

[3.3.1 Use the existing weighted sample data 11](#_Toc124596031)

[3.3.2 Generate a weighted sample data 13](#_Toc124596032)

[3.4 RUNNING 3A\_Distance\_Metrics.R 15](#_Toc124596033)

[Step 1: Read admin data 16](#_Toc124596034)

[Step 2: Obtain admin freq tables. 16](#_Toc124596035)

[Step 3: Merge admin + Weighted sample freq tables 17](#_Toc124596036)

[Step 4: Create domains of quality indicators 17](#_Toc124596037)

[Step 5: Compute distance metrics 18](#_Toc124596038)

[Step 6: Standardise distance metrics 18](#_Toc124596039)

[Step 7: Reshape, then tidy 18](#_Toc124596040)

[Step 8: Visualise the distance metrics 18](#_Toc124596041)

[3.5 RUNNING 3B\_R-indicator.R 19](#_Toc124596042)

[Step 1: Prepare an Auxiliary data file 20](#_Toc124596043)

[Step 2: Reshape and save benchmark data as row vectors 23](#_Toc124596044)

[Step 3: Declare variables in administrative data 24](#_Toc124596045)

[Step 4: Define macro variables 24](#_Toc124596046)

[Step 5: Compute R-indicators 25](#_Toc124596047)

[Step 6: Save in Excel and inspect 29](#_Toc124596048)

[Step 7: Visualising using scatterplots 30](#_Toc124596049)

[4. Troubleshooting Questions and Answers 31](#_Toc124596050)

[4.1 Questions and Answers 31](#_Toc124596051)

[How do I know where to customise the code to suit my needs? 31](#_Toc124596052)

[How to use Starting path in multiple machines? 31](#_Toc124596053)

[What are the commonly used commands? 31](#_Toc124596054)

[How to free up memory space and speed up RStudio? 31](#_Toc124596055)

[I get error messages when a pre-defined function is used. 31](#_Toc124596056)

[How do I modify pre-defined functions? 32](#_Toc124596057)

[What approaches are taken in programming? 33](#_Toc124596058)

[Can I ignore Warning messages? 33](#_Toc124596059)

[What version of R is used? 33](#_Toc124596060)

[4.2 Troubleshooting 33](#_Toc124596061)

[Unused argument error 33](#_Toc124596062)

[I get errors when computing… 33](#_Toc124596063)

[I am experiencing slowness in computation. 34](#_Toc124596064)

[Error: cannot allocate vector of size xxxx.x Gb 34](#_Toc124596065)

[References 34](#_Toc124596066)

[Citation 34](#_Toc124596067)

# Project aims and objectives

The Office for National Statistics (ONS) have strategic priorities on embedding and advancing the use of administrative data into their official statistics processes. Their immediate priority is to use administrative data in the quality assurance of the 2021 census and to develop the production of administrative-based population estimates (ABPEs). A more long-term priority is the Population Statistics Transformation Programme which will feed into a recommendation to Government due in 2023 on the future of census and population statistics. In particular, the objective is to create population characteristic estimates from administrative and integrated data sources.

Motivated by these initiatives, the University of Manchester was successfully awarded an ESRC grant from April 2021 to January 2023 titled: ‘Methodological Advancements on the Use of Administrative Data in Official Statistics’ (ES/V005456/1). The funded research enabled collaborations with the ONS on researching and developing methods to enhance the use of administrative data in official statistics.

This user manual concerns the sub-project related to developing a quality framework and quantitative measures to assess representativeness and coverage for a single administrative data source. We use univariate and bivariate distributions obtained from high- quality large random probability surveys (such as the UK Annual Population Survey) and compare them to distributions in the administrative data on a common set of variables based on distance metrics between these distributions. In addition, we developed a Representativity (R-) Indicator that is designed for quantifying the representativeness of population groups in the administrative data. Section 2 describes the methods underpinning the R-code hosted in GitHub and Section 3 describes the R-code with a running example of its outputs. We conclude in Section 4 with troubleshooting questions and answers.

# Quality indicators

In the R-code on GitHub we focus on quality indicators to identify errors arising from coverage and representativeness of a single administrative data source. It is vital that statistical agencies have good quality indicators to ensure the fit of administrative data to the population and to identify those sub-groups that are missing or over-covered, especially when the administrative data contributes to an integrated dataset for multisource processing or to quality assure other data sources, such as surveys and censuses.

## 2.1 Distance Metrics

In this section we compare distributions obtained from the administrative data with external population auxiliary information, either obtained directly from a census or estimated from a large probability-based random survey.

Denote variable having categories in the administrative dataset having M individuals. Let be the 0-1 indicator for individual being a member of category in variable *.* We can then calculate the counts for category : and note that We can also obtain the probability distribution of variable having category and calculate: . We note that the variable can also represent a cross-tabulation of two or more variables, for example age group sex.

Now assume we have equivalent estimates of these distributions from a census, or alternatively from a large probability-based random sample of size where every individual in the sample has an associated survey weight . The survey weights typically are calibrated to known population benchmarks and hence sum to the known population size . In this case, and . Moreover, the equivalent distribution is .

The entropy measures the uniformity of the probability distributions and hence can be compared when calculated on the administrative data distribution versus the population distribution. The formula for the entropy on the probability distribution is: - and similarly on the probability distribution

We can now use a variety of distance metrics to assess deviations between the distributions and . In this research, we assess three distance metrics: the Indicator of Dissimilarity (Duncan and Duncan, 1955), Hellinger’s Distance (HL) and the Kullback-Leibler divergence (KL). The formula for the three distance metrics is given in Table 1.

Table 1: Distance metrics between the distribution calculated from the administrative data and the distribution from the population on variable

|  |  |  |
| --- | --- | --- |
| Distance Metrics | Formula | Standarize |
| Indicator of Dissimilarity (ID) |  | 1-ID |
| Hellinger’s Distance (HL) |  | 1-HL |
| Kullback-Leibler Divergence (KL) |  | 1-KL |

There are subtle differences between these distance metrics. For example, Hellinger’s Distance places more weight on the smaller proportions compared to the larger proportions whereas the Indicator of Dissimilarity treats all proportions equally. More work is needed on standardizing the distance metrics into meaningful quality measures. We have yet to determine which distance metric should be used and therefore propose to include all of them in the R-code. This will facilitate more empirical work for future recommendations.

## 2.2 R-indicators

The R-indicator and its related partial R-indicators were originally designed to assess the representativeness of responses from a survey and are particularly useful as an objective function in the optimization of adaptive survey designs (Schouten, et al. 2009, Schouten and Shlomo, 2017). The R-indicators measure the contrast between those who are missing and not missing in the data and identify those groups that are not represented in the data. Here, we develop the R-indicator and partial R-indicators to assess the representativeness and coverage of an administrative dataset compared to a target population. Recent research by Bianchi, et al. (2019) adapts the R-indicator to the case where only population-based auxiliary information are available instead of sample-based frame information. We draw upon this research and utilize population-based auxiliary information where the population auxiliary information is obtained by weighted survey counts from a large probability-based random sample.

To calculate the population-based R-indicator, denote the response indicator equal to 1 for all units in the administrative dataset. We have information available on the values of a vector of auxiliary variables ***X***, for example, sex, age group, geographical region, ethnic minority group and employment status. Therefore, each is a binary indicator variable. We also assume that values of are observed for all individuals in the administrative dataset so that is observed.

Assume we know at the aggregate level: the population total and population cross-products . This information is known as the population–based auxiliary information. If this information is not available at the population level we can estimate the aggregates and cross-products using a large probability-based random sample, denoted , where each individual in the sample has a survey weight . The estimated population-based auxiliary information is then: and . We also know the overall total in the population, denoted by .

Response propensities are defined as the conditional expectation of the response indicator variable given the values of specified variables: In the population-based setting we model the response propensities under an identity link function where the true response propensities satisfy: , . For the linear probability model, the estimate of in the sample-based scenario is given by: , where is the design weight and denotes the sample under analysis.

In the case of population-based auxiliary information where we know both population totals and cross-products, we note that and are unbiased estimates for and , respectively and that in large samples we may expect that and . It follows that, in the population-based setting, we may approximate by and we refer to the propensities as ‘participation’ propensities. Note that is computed only on the set of individuals in the administrative data. In addition, in our setting of assessing the representativeness in administrative data, the design weight is the inverse of the coverage weight: where is the number of individuals in the administrative dataset.

In the population-based setting, an estimator for the R-indicator is given by where and is estimated as above. This estimator of the R-indicator makes the estimator linear in which provides an advantage for size bias adjustment computations (although given the large administrative datasets, a size bias adjustment is not needed). Furthermore, we use propensity weighting by to adjust for coverage bias. The R-indicator measures the variation of the sub-group participation propensities. If the participation propensities are all equal and there is no variation in sub-group participation, the R-indicator would obtain a value of 1.

The unconditional partial R-indicator measures the amount of variation of the participation propensities between the categories of a variable. The larger the between-category variation is, the stronger the relationship is and the stronger the impact of the variable on a lack of representativeness. As earlier, let be one of the components of the vector. The variable is categorical and assume it has categories. Let denote the weighted respondent size in category in the administrative data, for That means where is the 0-1 indicator for participating unit being a member of category and given the definition of as the inverse coverage weight. Define the average of the participation propensities in category of for the units in the administrative dataset and the overall average participation probability based on the estimated population-based participation propensities . The estimate for the unconditional partial R-indicator for variable is: . The upper bound of the unconditional partial R- indicator is 0.5. The larger the value of the partial R-indicator, the stronger the association of the variable with a lack of representativeness in the administrative dataset. By computing and comparing the unconditional partial indicators for a set of variables it can be established for which variables the relationships are strongest. The unconditional partial R-indicator at the category level for variable is and can assume positive and negative values. Note that at the category-level, a negative sign represents under-representation and a plus sign represents over-representation.

Finally, we note that when producing estimates from the administrative dataset, one should weight each individual by its inverse participation propensity: to adjust for coverage bias in the estimates.

# User Guide on the R-package

This package assumes that there is a Census file to obtain auxiliary population estimates and the administrative datasets under analysis. However, in real settings we would not have a Census microdata to work with, rather we would have a large probability-based survey sample for estimating population distributions. Therefore, we draw a random sample from the Census microdata to support this scenario and obtain auxiliary population totals from weighted sample counts.

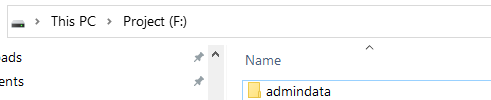
## 3.1 Download and inspect the contents

### 3.1.1 Download

Please visit the github site here: [qualadmin link](https://github.com/sook-tusk/qualadmin). Click on Code at the top-right corner. Then, click on Download ZIP to download to your local machine.

Now, the downloaded folder needs to be placed in the designated location. We recommend users decide the appropriate Drive (C, D, E, F, etc) to house the downloaded contents. Then, **create a new folder** called admindata in File Explorer of your PC. Users can customise the new folder name as appropriate. This is your **starting path**.

The screenshot showing **starting path**:



Under this Starting path, F:/admindata, place the downloaded folder from GitHub. Extract the zip folder as necessary.

As such, F:/admindata/qualadmin becomes the MASTER project folder. We’ll set it as working directory[[1]](#footnote-1) in RStudio later.

### 3.1.2 Downloaded contents explained

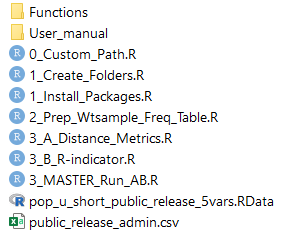
#### Example datasets

We provide two example data sources.

| Data type | File name |
| --- | --- |
| Administrative | public\_release\_admin.csv |
| Census | pop\_u\_short\_public\_release\_5vars.Rdata |

#### Folders and R scripts

Under F:\admindata\qualadmin folder, you’ll be presented with the following contents.



The **“User\_manual”** folder contains instructions on using the provided R code files.

The users do not need to do anything with the folder titled **“Functions”**. These pre-defined functions are used to either enclose complex procedures or perform repetitive tasks including cleaning and computing quality indicators. There are two files containing pre-defined functions. There is no need to run function files independently.

The functions will be automatically called in when the three main R script files are run: 2\_Prep\_Wtsample\_Freq\_Table.R 3\_A\_Distance\_Metrics.R 3\_B\_R-indicator.R

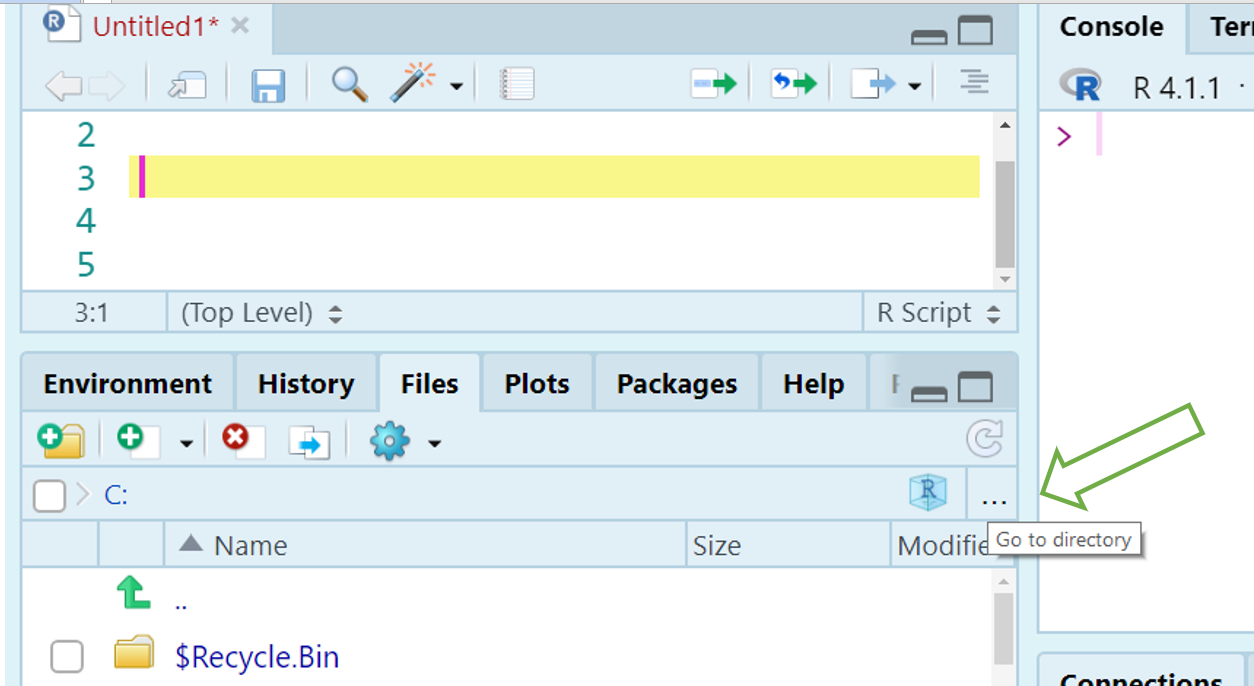
The 2\_Prep\_Wtsample\_Freq\_Table.R file creates necessary data needed to compute distance metrics and R-indicators. The master file, 3\_MASTER\_Run\_AB.R runs the above *two* main R script files, (3\_A\_Distance\_Metrics.R 3\_B\_R-indicator.R) automatically in sequence.

The first three files, 0\_Custom\_Path.R, 1\_Create\_Folders.R and 1\_Install\_Packages.R can be run to get ready to run the above main analysis files, as discussed in the following section.

## 3.2 Launch RStudio and get ready

### 3.2.1 Open the entire master folder in RStudio

First, launch RStudio. Then, we need to **open the entire folder** F:/admindata/qualadmin where downloaded materials are located.

Unfortunately, RStudio has no feature in the menu, but you could do so by accessing **Files** tab. Click on ... as shown below.  Then, locate the master folder. In our example, it is F:/admindata/qualadmin.

### 3.2.2 Set custom path

Click open the R script file, 0\_Custom\_Path.R. Customise the starting path as needed, and set the path to indicate the master folder. The example code is:

# Starting path (CUSTOMISE PLEASE)  
 setwd("F:/admindata")  
  
 # Master project folder (USE AS IT IS)  
 setwd("./qualadmin")  
  
 # Check your current directory  
 getwd()

Please ensure to use a single forward slash / as above. R will print an error when backward slash \ is used in path. For instance,

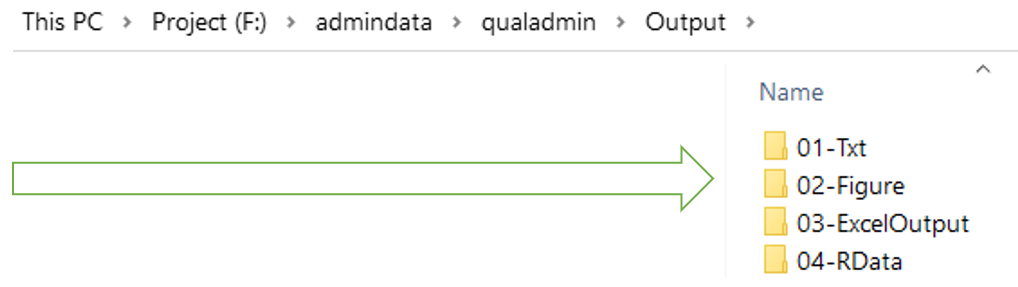
setwd("F:\admindata)`  
 Error: '\a' is an unrecognized escape in character string starting ""F:\a"

Please ensure your working directory is set at the master project path throughout the analytical steps.

### 3.2.3 Automatically create output folders

The three main R script files 2\_Prep\_Wtsample\_Freq\_Table.R, 3\_A\_Distance\_Metrics.R, 3\_B\_R-indicator.R produce outputs. The outputs may be text, figure or in spreadsheet form. For the existing programmes to work, users need to create dedicated output folders.

To do so, please click on the 1\_Create\_Folders.R file to open. Then run line by line. The resulting folder structure is provided here:



### 3.2.4 Install packages

The final preparation step is installing packages. Open 1\_Install\_Packages.R file, and run line by line.

#-----------------------------------  
 # Install packages (Run once)  
 #-----------------------------------  
   
 install.packages("ggplot2")  
 install.packages("tidyverse")  
   
 install.packages("car")

Now, you’re all set to proceed with quality measures indicators!

## 3.3 RUNNING 2\_Prep\_Wtsample\_Freq\_Table.R

This code file calculates the distributions of the weighted sample data. If the Census data or a weighted sample data are available, users consult 3.5.1 *Use the existing sample data*. For a scenario where these data are unavailable, users can generate the data as shown in 3.5.2 *Generate a weighted sample data*.

Open the 2\_Prep\_Wtsample\_Freq\_Table.R file.

The top of the code file concerns checking the current directory, reading in the pre-defined functions in the R environment and loading relevant R libraries.

getwd()

# Run the code file with functions.

source("Functions/1\_Functions.R")

# Define output file folders, path

fn\_output\_folder\_path()

# Disable scientific notation.

options(scipen = 999)

library("tidyverse") # data manipulation

library("ggplot2") # visualisation

library("janitor") # cross-tabulation

library("readxl") # read large csv file

library("writexl") # export to Excel

### 3.3.1 Use the existing weighted sample data

* Step 1: Read in the data.

df <- read\_csv("custom\_wtsample.csv")

dim(df) # obs = 1163650 (example)

glimpse(df) # Quick glance at the data

Users decide which variables are to be used for tabulation. For example, users may identify the following five variables.

names(df) # variable names

[1] "geog1" "sex" "agecode1" "eth\_code5"

[5] "econg"

* Step 2: Declare variables to be tabulated. Here, we declare all five variables using var object[[2]](#footnote-2). Users can customize the variable names here. By running the code below, R automatically saves the total number of variables, 5 in a macro called maxvar.

var <- c("geog1", "sex", "agecode1", # Please customise  
 "eth\_code5", "econg")

maxvar <- length(var) # No need to customise

maxvar

[1] 5

* Step 3: Obtain frequency table of categorical variables (count of categories).

This procedure is to assess and calculate *the distribution* of categories in the weighted sample. Users can run the pre-defined function, fn\_maxvar5\_freq\_table() to perform the task. The function automatically obtains counts and structure the output in long form, organised by each variable, and by its discrete category. In case of using four variables, users can use fn\_maxvar4\_freq\_table() instead[[3]](#footnote-3).

fn\_maxvar5\_freq\_table()

* Step 4: Carry out checks to see if the calculated frequency tables are accurate.

freq\_table[1:8, 1:9]

## seq twdigits n p oneway v by1 by2  
## 1 1 101 113250 0.09732308 1 1 geog1 01  
## 2 2 102 148400 0.12752976 1 1 geog1 02  
## 3 3 103 137450 0.11811971 1 1 geog1 03  
## 4 4 104 175100 0.15047480 1 1 geog1 04  
## 5 5 105 92300 0.07931938 1 1 geog1 05  
## 6 6 106 497150 0.42723327 1 1 geog1 06  
## 7 7 201 565350 0.48584196 1 2 sex 01  
## 8 8 202 598300 0.51415804 1 2 sex 02

When we printed the first 8 lines and 10 variables, we can see the count, n, and the corresponding proportion, p by each variable. The following code obtains the total observation size and confirms that the total proportion adds up to 1, for geog1 variable. Here, the total observation size can be viewed as the population size.

# Check whether the total adds up to 1  
 sum(freq\_table[1:6, "n"])

## [1] 1163650

sum(freq\_table[1:6, "p"])

## [1] 1

* Step 5: Rename and save.
* Step 6: Export the output frequency table in Excel with the file name, **Weightedsample\_freq\_table.xlsx**.
* Step 7: Save the R objects as RData. Done.

### 3.3.2 Generate a weighted sample data

* Step 1: Load a Census microdata. It is called **pop\_u\_short\_public\_release\_5vars** in the provided example code.

#H---------------------------------------  
 ##> 1. Load Census data  
 #H--------------------------------------  
  
 load("pop\_u\_short\_public\_release\_5vars.RData")  
 df <- pop\_u\_short\_public\_release\_5vars  
 dim(df) # obs = 1163659  
 names(df)

In our example Census data, we have 1,163,659 observations with five categorical variables including geography, sex, age groups, ethnic groups, and economic activity status. One can declare which variable to tabulate. Here, we declare all five variables using var object[[4]](#footnote-4).

var <- c("geog1", "sex", "agecode1",  
 "eth\_code5", "econg")

The description of categories, and distribution is shown below.

| Variable | Category | Description | N | (%) |
| --- | --- | --- | --- | --- |
| Total |  |  | 1163659 |  |
| geog1 | 1 | LA codes | 116128 | (10.0) |
|  | 2 | LA codes | 150139 | (12.9) |
|  | 3 | LA codes | 137520 | (11.8) |
|  | 4 | LA codes | 170624 | (14.7) |
|  | 5 | LA codes | 90873 | (7.8) |
|  | 6 | LA codes | 498375 | (42.8) |
| sex | 1 | male | 564905 | (48.5) |
|  | 2 | female | 598754 | (51.5) |
| agecode1 | 1 | 16-20 | 82426 | (7.1) |
|  | 2 | 21-25 | 94643 | (8.1) |
|  | 3 | 26-30 | 110296 | (9.5) |
|  | 4 | 31-35 | 120398 | (10.3) |
|  | 5 | 36-40 | 119393 | (10.3) |
|  | 6 | 14-45 | 101711 | (8.7) |
|  | 7 | 46-50 | 94209 | (8.1) |
|  | 8 | 51-55 | 100159 | (8.6) |
|  | 9 | 56-60 | 77799 | (6.7) |
|  | 10 | 61-65 | 65833 | (5.7) |
|  | 11 | 66-70 | 57305 | (4.9) |
|  | 12 | 71-75 | 51263 | (4.4) |
|  | 13 | 76-80 | 43678 | (3.8) |
|  | 14 | 81+ | 44546 | (3.8) |
| eth\_code5 | 1 | White | 1081812 | (93.0) |
|  | 2 | Mixed/Multiple ethnic groups | 10487 | (0.9) |
|  | 3 | Asian/Asian British | 46446 | (4.0) |
|  | 3 | Black/African/Caribbean/Black British | 16268 | (1.4) |
|  | 4 | Other ethnic group | 8646 | (0.7) |
| econg | 1 | In employment(FT, PT) | 689140 | (59.2) |
|  | 2 | Unemployed | 27744 | (2.4) |
|  | 3 | Out of workforce | 446775 | (38.4) |

Using this prior information on the population distribution (based on the Census), we can mimic the distribution in a random sample. See the next step.

* Step 2: Then, we draw a random sample 1:50.
* Step 3: From the randomly selected sample (1163659/50 = 23273), we then obtain frequency table of categorical variables (count of categories). Users can run the pre-defined function, fn\_maxvar5\_freq\_table() to perform the task. The function[[5]](#footnote-5) automatically obtains counts and structure the output in long form, organised by each variable, and by its discrete category.

fn\_maxvar5\_freq\_table()

This procedure is to assess and calculate *the distribution* of categories in a random sample (N = 23273). Based on the counts of the randomly selected sample, we multiply the counts by 50. One may wonder why we multiply. As we *reduced* the census sample by drawing a random sample by the 1:50 ratio, we need to *convert* the shrank sample back to the original size (with the priori distribution). This explains why we multiply by 50 (Weighted Sample N = 23273 \* 50 = 1163650). This completes the process of generating weighted sample survey data.

* Step 4: Carry out checks to see if the calculated frequency tables are accurate.

freq\_table[1:8, 1:9]

## seq twdigits raw\_n n p oneway v by1 by2  
## 1 1 101 2265 113250 0.09732308 1 1 geog1 01  
## 2 2 102 2968 148400 0.12752976 1 1 geog1 02  
## 3 3 103 2749 137450 0.11811971 1 1 geog1 03  
## 4 4 104 3502 175100 0.15047480 1 1 geog1 04  
## 5 5 105 1846 92300 0.07931938 1 1 geog1 05  
## 6 6 106 9943 497150 0.42723327 1 1 geog1 06  
## 7 7 201 11307 565350 0.48584196 1 2 sex 01  
## 8 8 202 11966 598300 0.51415804 1 2 sex 02

When we printed the first 8 lines and 10 variables, we can see the count, n, and the corresponding proportion, p by each variable. The following code obtains the total observation size and confirms that the total proportion adds up to 1, for geog1 variable. Here, the total observation size can be viewed as the population size.

# Check whether the total adds up to 1  
 sum(freq\_table[1:6, "n"])

## [1] 1163650

sum(freq\_table[1:6, "p"])

## [1] 1

* Step 5: Rename and save.
* Step 6: Export the output frequency table in Excel with the file name, **Weightedsample\_freq\_table.xlsx**.
* Step 7: Save the R objects as RData. Done.

## 3.4 RUNNING 3A\_Distance\_Metrics.R

To calculate distance metrics, we first obtain one-way, and two-way frequency tables of categorical variables, and calculates proportions of each sub-category by each data source. Previously, we dealt with frequency tables for the weighted sample data. Here, we calculate distributions of administrative data. Next, we combine master (benchmark) and admin freq tables. Then, we create domains for quality indicators, and finally compute distance metrics as part of quality indicators. We offer three different types of distance metrics. To allow comparison across the metrics, we standardise the calculations.

Users can also produce a summary table of three types of distance metrics, and visualise the results.

The preliminary step is to ensure we have benchmark data. Simply *source* the previous file as below to update it.

source("2\_Prep\_Wtsample\_Freq\_Table.R")

### Step 1: Read admin data

df <- read\_csv("public\_release\_admin.csv")

## Rows: 1033664 Columns: 6  
## -- Column specification --------------------------------------------------------  
## Delimiter: ","  
## dbl (6): person\_id, geog1, sex, agecode1, eth\_code5, econg  
##   
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

tail(df)

## # A tibble: 6 x 6  
## person\_id geog1 sex agecode1 eth\_code5 econg  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1033659 1 1 6 1 1  
## 2 1033660 6 1 6 1 1  
## 3 1033661 6 2 5 1 1  
## 4 1033662 5 1 12 1 3  
## 5 1033663 1 2 9 1 1  
## 6 1033664 6 2 9 1 1

The last six observations of the example admin data are shown above. As the person id goes up to 1033664, we can see there are 1033664 observations in admin data. As such, we declare all five variables for tabulations. If variables are already defined using var, users can skip this part.

var <- c("geog1", "sex", "agecode1",  
 "eth\_code5", "econg")

### Step 2: Obtain admin freq tables.

As mentioned in the previous section, we obtain frequency table of categorical variables (count of categories). Users can run the pre-defined function, fn\_maxvar5\_freq\_table() to perform the task. The function[[6]](#footnote-6) automatically obtains counts and structure the output in long form, organised by each variable, and by its discrete category.

Once the frequency tables are obtained, we rename the object as Admin\_f\_table\_one. Let’s inspect Admin\_f\_table\_one.

head(Admin\_f\_table\_one)

## seq twdigits admin\_n admin\_perc oneway v by1 by2 by3 by4 by5  
## 1 1 101 137993 0.1334989 1 1 geog1 01 oneway 0 0  
## 2 2 102 124051 0.1200110 1 1 geog1 02 oneway 0 0  
## 3 3 103 131176 0.1269039 1 1 geog1 03 oneway 0 0  
## 4 4 104 139867 0.1353119 1 1 geog1 04 oneway 0 0  
## 5 5 105 142304 0.1376695 1 1 geog1 05 oneway 0 0  
## 6 6 106 358273 0.3466049 1 1 geog1 06 oneway 0 0

tail(Admin\_f\_table\_one)

## seq twdigits admin\_n admin\_perc oneway v by1 by2 by3 by4 by5  
## 340 340 404501 8557 0.0082783187 2 4 eth\_code5 04 econg 5 01  
## 341 341 404502 791 0.0007652390 2 4 eth\_code5 04 econg 5 02  
## 342 342 404503 5007 0.0048439338 2 4 eth\_code5 04 econg 5 03  
## 343 343 405501 3867 0.0037410609 2 4 eth\_code5 05 econg 5 01  
## 344 344 405502 255 0.0002466953 2 4 eth\_code5 05 econg 5 02  
## 345 345 405503 3420 0.0033086187 2 4 eth\_code5 05 econg 5 03

### Step 3: Merge admin + Weighted sample freq tables

fn\_merge\_one\_admin\_wtsample\_f\_table\_temp()

The code above merges two data sources.

### Step 4: Create domains of quality indicators

fn\_create\_domain\_temp()

To check the domains, we can use *Janitor* package’s tabyl function[[7]](#footnote-7). The function creates 15 domains, including five single variables’ domain, and ten bivariate domains.

display\_domain %>% tabyl(fct\_domain)

## fct\_domain n percent  
## geog1 6 0.017391304  
## sex 2 0.005797101  
## agecode1 14 0.040579710  
## eth\_code5 5 0.014492754  
## econg 3 0.008695652  
## geog1:sex 12 0.034782609  
## geog1:agecode1 84 0.243478261  
## geog1:eth\_code5 30 0.086956522  
## geog1:econg 18 0.052173913  
## sex:agecode1 28 0.081159420  
## sex:eth\_code5 10 0.028985507  
## sex:econg 6 0.017391304  
## agecode1:eth\_code5 70 0.202898551  
## agecode1:econg 42 0.121739130  
## eth\_code5:econg 15 0.043478261

### Step 5: Compute distance metrics

Run the functions to compute three types of distance metrics.

fn\_unstd\_distance\_metrics\_full()  
 fn\_unstd\_distance\_metrics\_tidy()

### Step 6: Standardise distance metrics

### Step 7: Reshape, then tidy

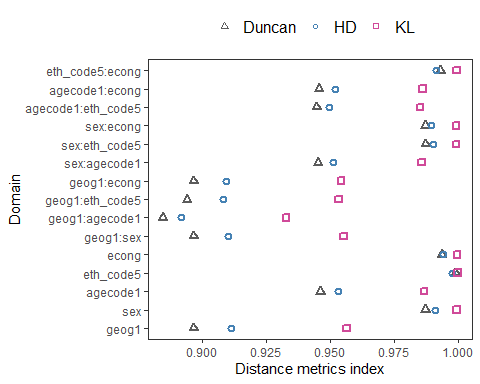
From wide form, the outputs have been reshaped to long form. Then, we keep standardised solutions. The results are as follows:

df <- distance\_metrics\_long  
 # std\_test(1-Duncan, 1-HD, 1-KL) only  
 df <- df %>% filter(std\_test\_use == 1)  
  
 df[1:9, c(1:2, 4:5, 9)]

## # A tibble: 9 x 5  
## domain\_id domain indicator index std\_test\_use  
## <int> <chr> <chr> <dbl> <dbl>  
## 1 1 geog1 Std\_Duncan 0.897 1  
## 2 1 geog1 Std\_t\_HD 0.911 1  
## 3 1 geog1 Std\_t\_KL 0.957 1  
## 4 2 sex Std\_Duncan 0.987 1  
## 5 2 sex Std\_t\_HD 0.991 1  
## 6 2 sex Std\_t\_KL 1.00 1  
## 7 3 agecode1 Std\_Duncan 0.946 1  
## 8 3 agecode1 Std\_t\_HD 0.953 1  
## 9 3 agecode1 Std\_t\_KL 0.987 1

### Step 8: Visualise the distance metrics

plot(p)



## 3.5 RUNNING 3B\_R-indicator.R

The file computes the overall R-indicator. Users can also proceed with computing **partial** R-indicators by category level, and variable level. The procedure can be computationally extensive. This is noted in the relevant section, so that users can allow some time to execute the code.

On this example, we will prepare the sample and population distributions and calculate an R-indicator for 5 variables: geog1 (6), sex (2), agecode1 (14) and eth\_cod5 (5) and econg (3).

As part of administrative data preparation, each of the variables should have their categories numbered 1,2,3…. We will use these numbers instead of the original names of the categories because it will enable us to do loops through the data. This is to facilitate building design matrix using dummy variables.

Along with administrative data, we also need benchmark data from Census. Assuming users have no access to Census data, we replace Census with weighted sample counts. The auxiliary data file contains these weighted sample counts.

As such, we will read in both administrative and auxiliary data files separately and compute R-indicators using matrix syntax in R. Due to the complexity of the procedure, we provide defined functions that users can execute with ease in practice. Users can consult Functions/2\_Functions\_R-indicators.R file for more details on the algorithm and operationalisation.

### Step 1: Prepare an Auxiliary data file

From the frequency table generated by using the weighted sample, we will prepare an auxiliary data file to be used for R-indicator computation procedures. This auxiliary file is used as benchmark. From the previous steps, we identified the population size of 1,116,350 (popsize = 1163650) from the weighted sample.

For R-indicator calculations, we need to compute meanpop by variables which are geography, sex, age groups, ethnic groups, and economic activity status. Users can simply run the function, fn\_meanpop\_auxiliary() to achieve this goal.

load(file = "Output/04-RData/Weightedsample\_freq\_table.RData")

load(file="Output/04-RData/var.RData")

**fn\_meanpop\_auxiliary**()

Exerpts of the function is provided below to aid readers’ understanding.

# Compute meanpop  
 auxiliary <- freq\_table %>%  
 filter(oneway == 1) %>%  
 group\_by(by1) %>%  
 mutate(meanpop = n / popsize) %>%  
 ungroup() %>%  
 dplyr::select(seq, count = n, by1,  
 v, by2, meanpop, raw\_n)

We first remove two-way and keep the one-way frequency table only. Then, by variable-level (indicated by the variable, by1), we compute meanpop. As seen before, the count of each category is stored in n. The meanpop is obtained by dividing n by popsize. For example, the value of **meanpop** for first category of geog1 is calculated as 113250/1163650 = 0.0973, and the second category of geog1 is 148400/1163650 = 0.1275 and so on.

Let’s look at the intermediate Auxiliary data. We can see the count of each variable (by1) and the sub-category (by2) for five variables. The population size is indicated as 1163560 (shown in the first row, under count column).

print(auxiliary)

## seq count by1 v by2 meanpop raw\_n type  
## 1 1 1163650 total 0 00 1.000000000 0 wtsample  
## 2 2 113250 geog1 1 01 0.097323078 2265 wtsample  
## 3 3 148400 geog1 1 02 0.127529756 2968 wtsample  
## 4 4 137450 geog1 1 03 0.118119710 2749 wtsample  
## 5 5 175100 geog1 1 04 0.150474799 3502 wtsample  
## 6 6 92300 geog1 1 05 0.079319383 1846 wtsample  
## 7 7 497150 geog1 1 06 0.427233275 9943 wtsample  
## 8 8 565350 sex 2 01 0.485841963 11307 wtsample  
## 9 9 598300 sex 2 02 0.514158037 11966 wtsample  
## 10 10 84600 agecode1 3 01 0.072702273 1692 wtsample  
## 11 11 93300 agecode1 3 02 0.080178748 1866 wtsample  
## 12 12 111200 agecode1 3 03 0.095561380 2224 wtsample  
## 13 13 124250 agecode1 3 04 0.106776092 2485 wtsample  
## 14 14 118200 agecode1 3 05 0.101576935 2364 wtsample  
## 15 15 99950 agecode1 3 06 0.085893525 1999 wtsample  
## 16 16 95800 agecode1 3 07 0.082327160 1916 wtsample  
## 17 17 95600 agecode1 3 08 0.082155287 1912 wtsample  
## 18 18 78450 agecode1 3 09 0.067417179 1569 wtsample  
## 19 19 67600 agecode1 3 10 0.058093069 1352 wtsample  
## 20 20 57950 agecode1 3 11 0.049800198 1159 wtsample  
## 21 21 50650 agecode1 3 12 0.043526834 1013 wtsample  
## 22 22 42250 agecode1 3 13 0.036308168 845 wtsample  
## 23 23 43850 agecode1 3 14 0.037683152 877 wtsample  
## 24 24 1083250 eth\_code5 4 01 0.930907060 21665 wtsample  
## 25 25 10450 eth\_code5 4 02 0.008980364 209 wtsample  
## 26 26 45600 eth\_code5 4 03 0.039187041 912 wtsample  
## 27 27 16300 eth\_code5 4 04 0.014007648 326 wtsample  
## 28 28 8050 eth\_code5 4 05 0.006917888 161 wtsample  
## 29 29 691300 econg 5 01 0.594078976 13826 wtsample  
## 30 30 28250 econg 5 02 0.024277059 565 wtsample  
## 31 31 444100 econg 5 03 0.381643965 8882 wtsample

Now, we prepare the data to build design matrix using dummy variables. To do so, from the total number of categories for each variable, we need to remove the **last** category of each categorical variable (group\_by(by1)). The last category is defined by max(by2)and removed accordingly. The first row (seq == 1) is not subject to this procedure and kept in the data. As shown in the code below, we keep rows if lastcat == 0, while filtering out cases which are the last category. We then drop the indicator for the last category (dplyr::select(-c(lastcat\_, lastcat))). Here, we explicitly instruct R to use dplyr package to access select function[[8]](#footnote-8).

col\_auxiliary <- auxiliary %>%  
 group\_by(by1) %>%  
 mutate(  
 lastcat\_ = ifelse(by2 == max(by2), 1, 0),  
 lastcat = ifelse(seq == 1, 0, lastcat\_)  
 ) %>%  
 filter(lastcat == 0) %>%  
 dplyr::select(-c(lastcat\_, lastcat)) %>%  
 ungroup()

Notice that from 5 variables, geog1 (6), sex (2), agecode1 (14) and eth\_cod5 (5) and econg (3), we now have 1+(nvar-1) for each variable so here it is 1+5+1+13+4+3=26 rows. We create a macro, numcat to store this information on the total row.

nrow(col\_auxiliary)

## [1] 26

numcat <- nrow(col\_auxiliary)  
numcat

## [1] 26

To inspect the setup on dummy variables (to be created later), let’s produce a cross-tabulation by by1 and by2.

dummychk <- col\_auxiliary

dummychk %>% tabyl(by1, by2)

00 01 02 03 04 05 06 07 08 09 10 11 12 13

agecode1 0 1 1 1 1 1 1 1 1 1 1 1 1 1

econg 0 1 1 0 0 0 0 0 0 0 0 0 0 0

eth\_code5 0 1 1 1 1 0 0 0 0 0 0 0 0 0

geog1 0 1 1 1 1 1 0 0 0 0 0 0 0 0

sex 0 1 0 0 0 0 0 0 0 0 0 0 0 0

total 1 0 0 0 0 0 0 0 0 0 0 0 0 0

All look good. At this stage, we keep meanpop only, dropping other columns, and print meanpop in a single column (column vector). This column will be transposed before we save it as row vectors. Before reshaping the data, we carefully inspect the values of meanpop.

# Print the column vector, meanpop  
 print(col\_auxiliary)

## # A tibble: 26 x 1  
## meanpop  
## <dbl>  
## 1 1   
## 2 0.0973   
## 3 0.128   
## 4 0.118   
## 5 0.150   
## 6 0.0793   
## 7 0.486   
## 8 0.0727   
## 9 0.0802   
## 10 0.0956   
## 11 0.107   
## 12 0.102   
## 13 0.0859   
## 14 0.0823   
## 15 0.0822   
## 16 0.0674   
## 17 0.0581   
## 18 0.0498   
## 19 0.0435   
## 20 0.0363   
## 21 0.931   
## 22 0.00898  
## 23 0.0392   
## 24 0.0140   
## 25 0.594   
## 26 0.0243

### Step 2: Reshape and save benchmark data as row vectors

We use t(col\_auxiliary) to transpose and tidy the reshaped data. We then generate ttt as an indicator of benchmark data. We use this indicator for merging with the administrative data later. Save the data as popmean\_row\_vector.

# Transpose to arrange in row vector format.  
 temp <- t(col\_auxiliary)  
 temp <- as.data.frame(temp)  
 row.names(temp) <- 1:nrow(temp)  
  
# generate merge id, ttt.  
 temp$ttt <- 0  
  
# Rename variables "popmean1- popmean26"  
 names(temp) <- c(paste0("popmean", 1:numcat), "ttt")  
  
# SAVE  
 popmean\_row\_vector <- temp

Let’s see the row vector names, “popmean1- popmean26” and ttt.

names(popmean\_temp)

## [1] "popmean1" "popmean2" "popmean3" "popmean4" "popmean5" "popmean6"   
## [7] "popmean7" "popmean8" "popmean9" "popmean10" "popmean11" "popmean12"  
## [13] "popmean13" "popmean14" "popmean15" "popmean16" "popmean17" "popmean18"  
## [19] "popmean19" "popmean20" "popmean21" "popmean22" "popmean23" "popmean24"  
## [25] "popmean25" "popmean26" "ttt"

popmean\_temp[, 1:5]

## popmean1 popmean2 popmean3 popmean4 popmean5  
## 1 1 0.09732308 0.1275298 0.1181197 0.1504748

Now, our benchmark data preparation is completed.

### Step 3: Declare variables in administrative data

The next step is to declare variables to be used for computing the R-indicator. These variable names should be from the administrative data. As mentioned earlier, users may skip this part if earlier defined var is unchanged and consistent throughout benchmark data and administrative data. In the provided example file, we save var object and load it, rather than defining at each stage.

Follow the instructions if users need to declare here. Users can define their own variables and save as a macro, var. For instance, users may use four variables and define as below.

var <- c(“geog1”, “sex”, “agecode1”, “econg”)

For demonstration, we use five variables for R-indicator calculations and declared as such.

# Customise as needed.  
var <- c("geog1", "sex", "agecode1",  
 "eth\_code5", "econg")  
var

## [1] "geog1" "sex" "agecode1" "eth\_code5" "econg"

# End of custom variables.

This concludes steps 1 through 3. Users can customise some setups up to this point.

### Step 4: Define macro variables

Along with the defined variables, we need to ensure macro variables are generated correctly. These macro variables are not designed to customise and designed to run without altering the code.

var

variablenum

maxvar

popsize

The total number of variables and the total number of categories of each categorical variables will be stored in a macro called variablenum and maxvar. We use var macro, as we defined earlier, to derive these two macros.

variablenum <- length(var) # No need to customise  
maxvar <- length(var) # No need to customise  
  
variablenum ; maxvar

## [1] 5

## [1] 5

Earlier, we also generated popsize to feed the information on the (total observation) size of the the benchmark data to the programme. The remaining macros, such as respop, piinv and rrate, will be automatically generated by the pre-defined functions.

respop

piinv

rrate

respop is the (sample) size of the administrative data. piinv refers to the inverse pi. rrate is computed by resppop/popsize.

### Step 5: Compute R-indicators

Once steps 1 through 4 are completed, we are set to carry out computing R-indicators. These procedures are automated via pre-defined functions using complex matrix and data management syntax. Please note that some procedures are computationally intensive.

We first open the corresponding administrative data, and notice the number of rows is 1,033,664.

aa <- read\_csv("public\_release\_admin.csv")

## Rows: 1033664 Columns: 6  
## -- Column specification --------------------------------------------------------  
## Delimiter: ","  
## dbl (6): person\_id, geog1, sex, agecode1, eth\_code5, econg  
##   
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

nrow(aa) # 1033664

## [1] 1033664

We need several empty objects to hold data to get the functions to work as intended. Note that we have five functions that permit us to compute overall R-indicators, along with additional functions for obtaining partial R-indicators. Each function will generate objects including vvv, pop\_respmean, des\_pop\_respmean, gh, and R\_indicators as indicated in the function name. The partial R-indicators can be found in the data object, partial.

df <- NULL  
 between <- NULL  
 partial <- NULL  
 partialtemp <- NULL  
 fn\_r\_indicator\_1\_vvv()  
 fn\_r\_indicator\_2\_pop\_respmean()  
 fn\_r\_indicator\_3\_des\_pop\_respmean()  
 fn\_r\_indicator\_4\_gh()  
 fn\_R\_indicators()  
   
 # Partial R-indicators  
 fn\_r\_indicator\_partialtemp()  
 fn\_r\_indicator\_domain\_order\_partial()

Let’s go over one function at a time.

The utility of fn\_overall\_r\_indicator\_1\_vvv() is to build design matrix. Starting with ensuring that the admin data only contains the declared variables and factorise them, the function defines macro variables, resppop and rrate using the data object, aa. To prepare for design matrix, the code creates dummy variables, using **fastDummies** R library package. Once all dummy variables generated, we need to remove the last category. As such, the function detects the categories of each variable and drops the last category.

Let’s see the vvv object, which contains the design matrix with weights. The weights are calculated by the inverse of *rrate* (finalwgt = 1/rrate). The last five columns are printed below.

# fn\_overall\_r\_indicator\_1\_vvv()  
  
 from <- ncol(vvv)-4  
 vvv[1:8, from: ncol(vvv)]

## # A tibble: 10 x 5  
## des24 des25 des26 finalwgt piinv  
## <int> <int> <int> <dbl> <dbl>  
## 1 0 1 0 1.13 1  
## 2 0 1 0 1.13 1  
## 3 0 1 0 1.13 1  
## 4 0 1 0 1.13 1  
## 5 0 1 0 1.13 1  
## 6 0 0 0 1.13 1  
## 7 0 0 0 1.13 1  
## 8 0 1 0 1.13 1

The fn\_overall\_r\_indicator\_2\_pop\_respmean() prepares the distributions of the administrative data. Using the data object, vvv, we obtain weighted sample counts and produces a row vector called *respmean\_row\_vector*. Merging the corresponding popmean\_row\_vector from the benchmark data we prepared at the step 2 earlier, the function combines both mean vectors from the two data sources.

The combined mean vectors are stored at the pop\_respmean data object. We will inspect the last five columns.

# fn\_overall\_r\_indicator\_2\_pop\_respmean()  
  
 from <- ncol(pop\_respmean)-4  
 pop\_respmean[, from: ncol(pop\_respmean)]

## respmean25 respmean26 finalwgt piinv ttt  
## 1 0.5896214 0.02252997 1.125753 1 0

The function, fn\_overall\_r\_indicator\_3\_des\_pop\_respmean(), allows us to combine the design matrix with the pop\_respmean. We store the data at des\_pop\_respmean.

# fn\_overall\_r\_indicator\_3\_des\_pop\_respmean()  
  
 names(des\_pop\_respmean)

## [1] "des1" "des2" "des3" "des4"   
## [5] "des5" "des6" "des7" "des8"   
## [9] "des9" "des10" "des11" "des12"   
## [13] "des13" "des14" "des15" "des16"   
## [17] "des17" "des18" "des19" "des20"   
## [21] "des21" "des22" "des23" "des24"   
## [25] "des25" "des26" "finalwgt" "piinv"   
## [29] "popmean1" "popmean2" "popmean3" "popmean4"   
## [33] "popmean5" "popmean6" "popmean7" "popmean8"   
## [37] "popmean9" "popmean10" "popmean11" "popmean12"   
## [41] "popmean13" "popmean14" "popmean15" "popmean16"   
## [45] "popmean17" "popmean18" "popmean19" "popmean20"   
## [49] "popmean21" "popmean22" "popmean23" "popmean24"   
## [53] "popmean25" "popmean26" "respmean1" "respmean2"   
## [57] "respmean3" "respmean4" "respmean5" "respmean6"   
## [61] "respmean7" "respmean8" "respmean9" "respmean10"   
## [65] "respmean11" "respmean12" "respmean13" "respmean14"   
## [69] "respmean15" "respmean16" "respmean17" "respmean18"   
## [73] "respmean19" "respmean20" "respmean21" "respmean22"   
## [77] "respmean23" "respmean24" "respmean25" "respmean26"   
## [81] "seq" "responsesamp1"

from <- ncol(des\_pop\_respmean)-4  
 des\_pop\_respmean[1:6, from: ncol(des\_pop\_respmean)]

## # A tibble: 6 x 5  
## respmean24 respmean25 respmean26 seq responsesamp1  
## <dbl> <dbl> <dbl> <int> <dbl>  
## 1 0.0139 0.590 0.0225 1 1  
## 2 0.0139 0.590 0.0225 2 1  
## 3 0.0139 0.590 0.0225 3 1  
## 4 0.0139 0.590 0.0225 4 1  
## 5 0.0139 0.590 0.0225 5 1  
## 6 0.0139 0.590 0.0225 6 1

In the fn\_overall\_r\_indicator\_4\_gh(), we compute the difference from the mean vectors and the weight variables. Exerpts of the code from the function below show that the differences are stored in rsam and psam. By adding *rsam* and *psam* to the intermediate data object, des\_pop\_respmean, we obtain the gh data. This concludes the pre-matrix preparation part.

# use df for programming.

df <<- data.frame(des\_pop\_respmean)

# Prep for loop.

des\_col <<- c(paste0("des" , 1:numcat))

respmean\_col <<- c(paste0("respmean", 1:numcat))

popmean\_col <<- c(paste0("popmean" , 1:numcat))

des <<- df[, des\_col]

respmean <<- df[, respmean\_col]

popmean <<- df[, popmean\_col]

rsam <<- des - respmean

psam <<- des - popmean

temp <<- data.frame(rsam, psam)

# Rename variables

colnames(temp) <<- c(paste0("rsam", 1:numcat),

paste0("psam", 1:numcat))

# Combine

gh <<- cbind(des\_pop\_respmean, temp)

The fn\_R\_indicators() uses matrix syntax to calculate propensity scores, prior to computing R-indicators. We calculate two kinds of propensity scores – one that used only population information (roipop, or prop\_pop) and the other which used a mixture of the response data and the population information (roimix, or prop\_mix).

In terms of partial R-indicators, we demonstrate using *prop\_mix*. Running fn\_r\_indicator\_partialtemp() users can yield partial R-indicators at the variable level and at the category level. The function, fn\_r\_indicator\_domain\_order\_partial() helps us to organise domains.

fn\_r\_indicator\_partialtemp()  
 fn\_r\_indicator\_domain\_order\_partial()  
 # View(partial)

### Step 6: Save in Excel and inspect

At this stage, users can inspect the output accordingly. Let’s have a look. Here, we can see the overall R-indicator is estimated as 0.496 based the administrative data (N=1033664). Looking at the variable-level R-indicator (see rows 4-8), geog1 was seen to have the greatest R-indicator (0.04) compared to econg (0.0002).

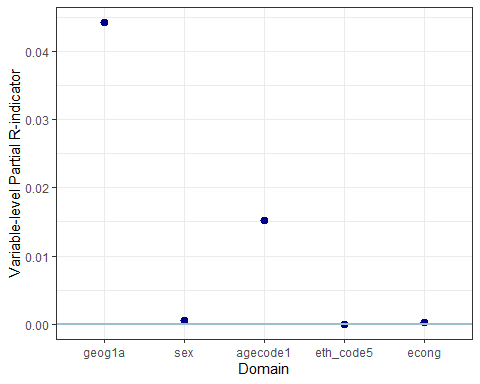
partial[1:17, c(1:2, 4, 8:10)]

## seq domain R\_indicator count n\_cat domain\_n  
## 1 1 Overall 0.4960263148 NA <NA> NA  
## 2 2 mrphatall 0.9597768609 NA <NA> NA  
## 3 3 resppop 1033664.0000000000 NA <NA> NA  
## 4 4 geog1 0.0442418275 NA <NA> NA  
## 5 5 sex 0.0004983014 NA <NA> NA  
## 6 6 agecode1 0.0152841196 NA <NA> NA  
## 7 7 eth\_code5 0.0000296805 NA <NA> NA  
## 8 8 econg 0.0002117090 NA <NA> NA  
## 9 9 des1 NA 0 1 0  
## 10 10 geog1\_1 0.0879501347 137993 1 1  
## 11 11 geog1\_2 -0.0192794816 124051 2 1  
## 12 12 geog1\_3 0.0219039079 131176 3 1  
## 13 13 geog1\_4 -0.0366161292 139867 4 1  
## 14 14 geog1\_5 0.1396946734 142304 5 1  
## 15 15 geog1\_6 -0.1216543425 358273 6 1  
## 16 16 sex\_1 -0.0162005503 489228 1 2  
## 17 17 sex\_2 0.0153571986 544436 2 2

### Step 7: Visualising using scatterplots

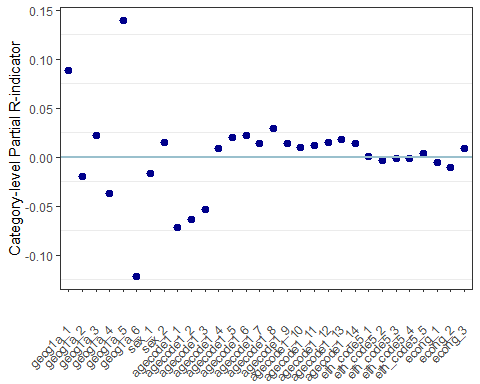
The visualisation of R-indicator by the variable level is shown as an example:

plot(p1)



And for R-indicator by the category-level:

plot(p2)



This concludes the manual. Thank you for taking the time reading the material. Please get in touch with any query or errata at [fanfurcada@gmail.com](mailto:fanfurcada@gmail.com).

If you need technical support, please consult the following Troubleshooting Q & A section.

# 4. Troubleshooting Questions and Answers

## 4.1 Questions and Answers

### How do I know where to customise the code to suit my needs?

Unless indicated as “Customise as needed”, users can run the code as it is. Please consult each code file.

### How to use Starting path in multiple machines?

If users plan to use different machines, simply by changing the “starting path”, users can carry out the analysis with minimal disruption. To achieve this, please ensure to use the consistent master project folder name.

### What are the commonly used commands?

Most commonly used commands in the tidyverse package are:

arrange : sort variables.  
 bind\_rows: append multiple dataframes.  
 mutate : manipulate variables, and  
 create new variables based on old variables.  
 select : order, and keep(drop) variables of interest.  
 shell.exec: launch a software and opens the target file (Windows PC only)

### How to free up memory space and speed up RStudio?

You can remove objects that you no longer need.

*# To remove objects except for certain objects*

ls()

keepobjectslist <- c("a", "b", "c")

rm(list = ls()[!ls() %in% keepobjectslist])

ls()

### I get error messages when a pre-defined function is used.

Users can inspect the codes used in the function, and identify the issues. It is recommended NOT edit the function file directly, as the functions are used repeatedly, and the interlinked sections may not run as expected. Where preferable, users may copy the codes in the function, and use locally with minor tweaks.

### How do I modify pre-defined functions?

Users can modify 1\_Functions and 2\_Functions\_R-indicators.R under *Functions* folder.

# 1\_Functions.R  
fn\_output\_folder\_path <- function() {  
  
 currentdate <<- Sys.Date()  
 txtpath <<- "Output/01-Txt/"  
 figpath <<- "Output/02-Figure/"  
 xlsxpath <<- "./Output/03-ExcelOutput/"  
 Rdatapath <<- "Output/04-RData/"  
}

We can check how the output folder names are set as path to save the results during the analytical process.

fn\_output\_folder\_path()

Let’s run the function. We can see that xlsxpath is set as "./Output/03-ExcelOutput/".

xlsxpath

## [1] "./Output/03-ExcelOutput/"

Let’s customise the xlsxpath, by renaming the folder name. If we customise 1\_Functions.R file, we can edit the information enclosed in the brackets. Notice that we use <<- with functions so that the object created by a function will exist in the global R environment. This is very important.

Alternatively, we could ignore the pre-defined function and just write relevant lines of code and keep it in the main R script file. For instance, we could put output\_folder\_path at the top of the 2\_Prep\_Wtsample\_Freq\_Table.R. Here, we edited the xlsxpath. Notice that fn\_output\_folder\_path <- function() { } is removed.

xlsxpath\_2 <- "./Output/03-Excel/"  
   
 xlsxpath\_2

## [1] "./Output/03-Excel/"

#H---------------------------------------  
 ## > Step 1. Load Census data  
 #H--------------------------------------  
 # load("pop\_u\_short\_before\_sim\_5vars.RData")

Notice that we use <-. Using <<- is not necessary here. Users can remember the usage of <- and can modify the functions as appropriate, should the function incur errors.

### What approaches are taken in programming?

When loop is used, base R functions were used (table, tapply, etc). For data manipulation, tidyverse package was used extensively. This strategy is partly to improve readability of the code.

To enhance users’ workflow, output files are programmed to launch using the pre-defined functions.

### Can I ignore Warning messages?

Some packages alert users with compatibility issues arising from old version. These can be ignored. For example,

library("fastDummies")  
 Warning message:  
 package 'fastDummies' was built under  
 R version 4.1.2  
  
 library(rlist)  
 Warning message:  
 package 'rlist' was built under R version 4.1.2

### What version of R is used?

Tested with Windows PC. R version used: 4.1.1 RStudio version: RStudio 2022.12.0 Build 353.

## 4.2 Troubleshooting

### Unused argument error

For example, sim %>% select(geog1) the select command can cause an error:

Error in select(., geog1) :  
 unused arguments (geog1)

This maybe due to the conflict in packages.

The error can be fixed by adding the name of the package used, dplyr, explicitly. sim %>% dplyr::select(geog1)

### I get errors when computing…

Please inspect zero cells, and ensure 0 (numeric value) is entered for n and perc, as well as admin\_n and admin\_perc. Errors may occur with NA coding and data attributes(character, factor, numeric).

### I am experiencing slowness in computation.

R can be not responsive if memory is full. Please identify bottlenecks and remove them. It may be due to certain commands. For example, View(object) command could take a while if the object is huge in size. Unless one should inspect the data, suppress the View command to expedite the computation where possible.

It can also be the case that for loop functions can be slow as well. In some instances, removing objects may help as this procedure can free up memory space. See above commonly used commands for more information.

### Error: cannot allocate vector of size xxxx.x Gb

If matrix symbols have entered mistakenly, R shows an error message like this. Please double check whether there are any mistakes. For instance, one may have typed a\*b instead of a%\*%b. Users can type memory.limit() to check the current memory limit and increase as necessary.

# References

Bianchi, Annamaria, Natalie Shlomo, Barry Schouten, Damião N. Da Silva, and Chris Skinner. ‘Estimation of Response Propensities and Indicators of Representative Response Using Population-Level Information’. Survey Methodology 45 (2) (2019): 217–47.

Duncan, Otis Dudley, and Beverly Duncan. ‘A Methodological Analysis of Segregation Indexes’. American Sociological Review 20, no. 2 (1955): 210–17. <https://doi.org/10.2307/2088328.>

R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

Schouten, Barry, Fannie Cobben, Jelke Bethlehem. ‘Indicators for the Representativeness of Survey Response.. Survey Methodology 35(1) (2009): 101 – 113.

Schouten, Barry, and Natalie Shlomo. ‘Selecting Adaptive Survey Design Strata with Partial R-indicators’. International Statistical Review 85(1) (2017): 143-163.

# Citation

Please cite this work as:

Kim, Sook & Shlomo, Natalie (2023). “Quality Indicators for Administrative Data User Manual For R Package on GitHub, version 1.2”, available at <https://github.com/sook-tusk/qualadmin>

1. Notice that the terms, folder, directory, and path are used interchangeably in the user manual. [↑](#footnote-ref-1)
2. As the var object is treated as a global macro, the programme runs automatically using the information stored in global macro, and produces the results. Users can save var in a separate RData file and load it in each session, instead of repeating the procedure. [↑](#footnote-ref-2)
3. We do not provide functions for the maximum variable size beyond five. In such cases, users can create their own functions by consulting the provided functions. [↑](#footnote-ref-3)
4. As the var object is treated as global macro, the programme runs automatically using the information stored in global macro, and produces the results. [↑](#footnote-ref-4)
5. We also provide fn\_maxvar4\_freq\_table() for users who declare four categorical variables. We do not provide functions for the maximum variable size beyond five. In such cases, users can create their own functions by consulting the provided functions. [↑](#footnote-ref-5)
6. We also provide fn\_maxvar4\_freq\_table() for users who wish to declare four categorical variables. [↑](#footnote-ref-6)
7. This is essentially almost identical to table(display\_domain$fct\_domain), but the approach by tabyl produces percent by default. [↑](#footnote-ref-7)
8. This is to avoid warning messages from R when R searches for a particular function from two different packages. [↑](#footnote-ref-8)