May 2025

Quality Indicators for Administrative Data[1]

User Manual For R Package on GitHub

Sook Kim and Natalie Shlomo
University of Manchester

Version 3.0

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

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.

2. 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 v having categories $h, h = 1, 2 \dots H$ in the administrative dataset having M individuals. Let $\Delta_{h,i}^{v}$ be the 0-1 indicator for individual i being a member of category h in variable v. We can then calculate the counts for category h : $m_h^v = \sum_{i=1}^{M} \Delta_{h,i}^{v}$ and note that $\sum_{h=1}^{H} m_h^v = M$. We can also obtain the probability distribution of variable v having category $h, h = 1, 2 \dots H$ and calculate: $p_h^v = \frac{m_h^v}{M}$. We note that the variable v 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 n where every individual i in the sample has an associated survey weight w_i . The survey weights typically are calibrated to known population benchmarks and hence sum to the known population size N. In this case, $n_h^v = \sum_{i=1}^{n} w_i \Delta_{h,i}^{v}$ and $\sum_{h=1}^{H} n_h^v = N$. Moreover, the equivalent distribution is $q_h^v = \frac{n_h^v}{N}$.

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 $\{p_h^v, h = 1, \dots, H\}$ is: $-\sum_h p_h^v \, log(p_h^v)$ and similarly on the probability distribution $\{q_h^v, h = 1, \dots, H\}$.

We can now use a variety of distance metrics to assess deviations between the distributions $\{p_h^v, h = 1, \dots, H\}$ and $\{q_h^v, h = 1, \dots, H\}$. 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 $\{p_h^v, h = 1, \dots, H\}$ and the distribution from the population $\{q_h^v, h = 1, \dots, H\}$ on variable v

Distance Metrics	Formula	Standarize
Indicator of Dissimilarity (ID)	$\frac{1}{2}\sum_h \lvert p_h - q_h \rvert$	1-ID
Hellinger's Distance (HL)	$\frac{1}{\sqrt{2}}\sqrt{\sum_h \left(\sqrt{p_h} - \sqrt{q_h}\right)^2}$	1-HL

Kullback-Leibler Divergence (KL)	$\sum_h p_k \, log\left(\frac{p_h}{q_h}\right)$	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 r_i equal to 1 for all units in the administrative dataset (denoted by 'A'). We have information available on the values $\boldsymbol{x_i} = (x_{1,i}, x_{2,i}, \dots, x_{K,i})^T$ of a vector of K auxiliary variables $\boldsymbol{X}$, for example, sex, age group, geographical region, ethnic minority group and employment status. Therefore, each $x_{k,i}$ is a binary indicator variable. We also assume that values of $\boldsymbol{x_i}$ are observed for all individuals in the administrative dataset so that $\{x_i; i \in A\}$ is observed.

Assume we know $\boldsymbol{x_i}$ at the aggregate level: the population total $\sum_U \boldsymbol{x_i}$ and population cross-products $\sum_U \boldsymbol{x_i}\,\boldsymbol{x_i^T}$. 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 s, where each individual i in the sample has a survey weight w_i. The estimated population-based auxiliary information is then: $\sum_s w_i \boldsymbol{x_i}$ and $\sum_s w_i \boldsymbol{x_i x_i^T}$. We also know the overall total in the population, denoted by N.

Response propensities are defined as the conditional expectation of the response indicator variable r_i given the values of specified variables: $\rho_i \equiv \rho_X(\boldsymbol{x_i})$. In the population-based setting

we model the response propensities under an identity link function where the true response propensities satisfy: $\rho_i = \boldsymbol{x_i^T \beta}$, $i \in U$. For the linear probability model, the estimate of ρ_i in the sample-based scenario is given by: $\hat{\rho}_i^{OLS} = \boldsymbol{x_i^T}(\sum_{\boldsymbol{A}} d_i \, \boldsymbol{x_i x_i^T})^{-1} \sum_{\boldsymbol{A}} d_i \, \boldsymbol{x_i} r_i \;\; , i \in s'$, where d_i is an unknown design weight and A denotes the administrative data under analysis.

We approximate $\hat{\rho}_i^{OLS}$ by $\hat{\rho}_i^P = \boldsymbol{x_i^T}(\sum_{\boldsymbol{U}} \boldsymbol{x_i x_i^T})^{-1} \sum_{\boldsymbol{A}} d_i \boldsymbol{x_i}$, $i \in A$ and we refer to the propensities as 'participation' propensities. Note that $\hat{\rho}_i^P$ is computed only on the set of individuals in the administrative data. In addition, in our setting of assessing the representativeness in administrative data, we define an overall pseudo design weight d as the inverse of the coverage weight: $d = [M/N]^{-1}$ where M is the number of individuals in the administrative dataset.

In the population-based setting for administrative data, an estimator for the R-indicator is given by $\hat{R}_{\hat{\rho}^P} = 1 - \hat{S}_{\hat{\rho}^P}$ where $\hat{S}^2_{\hat{\rho}^P} = \frac{N}{N-1}\{\frac{1}{N}\sum_r d_i \hat{\rho}_i^P - [\frac{1}{N}\sum_r d_i]^2\}$ and $\hat{\rho}_i^P$ is estimated as above. Furthermore, we use propensity weighting by $\hat{\rho}_i^{P^{-1}}$ 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 $\boldsymbol{x_k}$ be one of the components of the vector $\boldsymbol{X}$. The variable $\boldsymbol{x_k}$ is categorical and assume it has H categories. Let m_h denote the weighted respondent size in category h in the administrative data, for $h = 1, 2, \ldots, H$. That means $m_h = \sum_{i \epsilon r} d\Delta_{h,i}$ where $\Delta_{h,i}$ is the 0-1 indicator for participating unit i being a member of category h and $\sum_{h=1}^H m_h = N$ given the definition of d as the inverse coverage weight. Define $\hat{\bar{\rho}}_h$ the average of the participation propensities in category h of $\boldsymbol{x_k}$ for the units in the administrative dataset and $\hat{\bar{\rho}}$ the overall average participation probability based on the estimated population-based participation propensities $\hat{\rho}_i^P$. The estimate for the unconditional partial R-indicator for variable $\boldsymbol{x_k}$ is: $R_U(\boldsymbol{x_k}) = \sqrt{\frac{1}{N}\sum_{h=1}^H m_h(\hat{\bar{\rho}}_h - \hat{\bar{\rho}})^2}$. 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 h for variable $\boldsymbol{x_k}$ is $R_U(\boldsymbol{x_k^h}) = \sqrt{\frac{m_h}{N}}\,(\hat{\bar{\rho}}_h - \hat{\bar{\rho}})$ 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 i by its inverse participation propensity: $\hat{\rho}_i^{P^{-1}}$ to adjust for coverage bias in the estimates.

The theory described here is when the administrative data is in the form of individual-level microdata. If the administrative data is very large, we can first aggregate it to a table spanned by the variables of interest (covariates) in the model prior to calculating the R-indicator and partial indicators. The description of the R-code for microdata-level administrative data is described in Section 3.5 and the description of the R-code for tabular-level administrative data in described in Section 3.6.

3. 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.

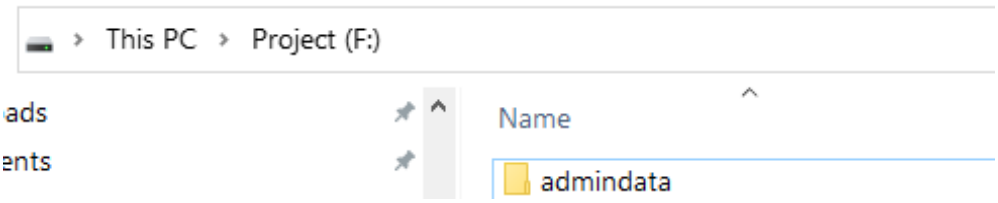
3.1 Download and inspect the contents

3.1.1 Download

Please visit the github site here: qualadmin link. 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] in RStudio later.

3.1.2 Downloaded contents explained

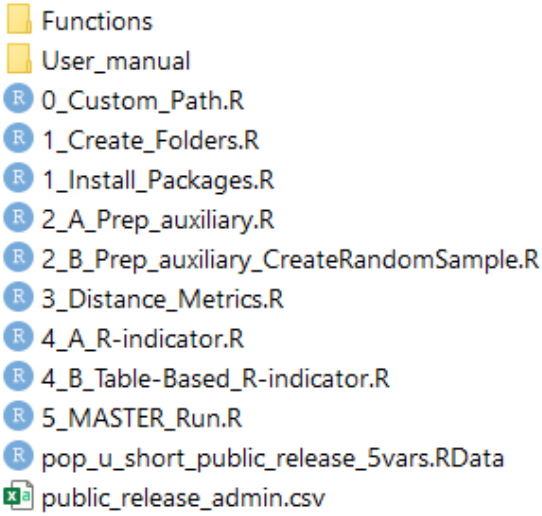
Example datasets

We provide two example data sources.

Data type	File name
Census	pop_u_short_public_release_5vars.Rdata
Administrative	public_release_admin.csv

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 is no need to run this function file independently.

The functions will be automatically called in when relevant R script files are run.

The auxiliary data generation files include `2_Prep_auxiliary.R` and `2_B_Prep_auxiliary_CreateRandomSample.R`. The two script files create necessary data needed to compute distance metrics and R-indicators. To obtain distrance metrics and R-

[1] Notice that the terms, folder, directory, and path are used interchangeably in the user manual.

indicators, user can run individual files including `3_A_Distance_Metrics.R, 4_A_R-indicator.R` and `4_B_Table-Based_R-indicator.R`. The master file, `5_MASTER_Run.R` runs all the above R script files automatically in sequence.

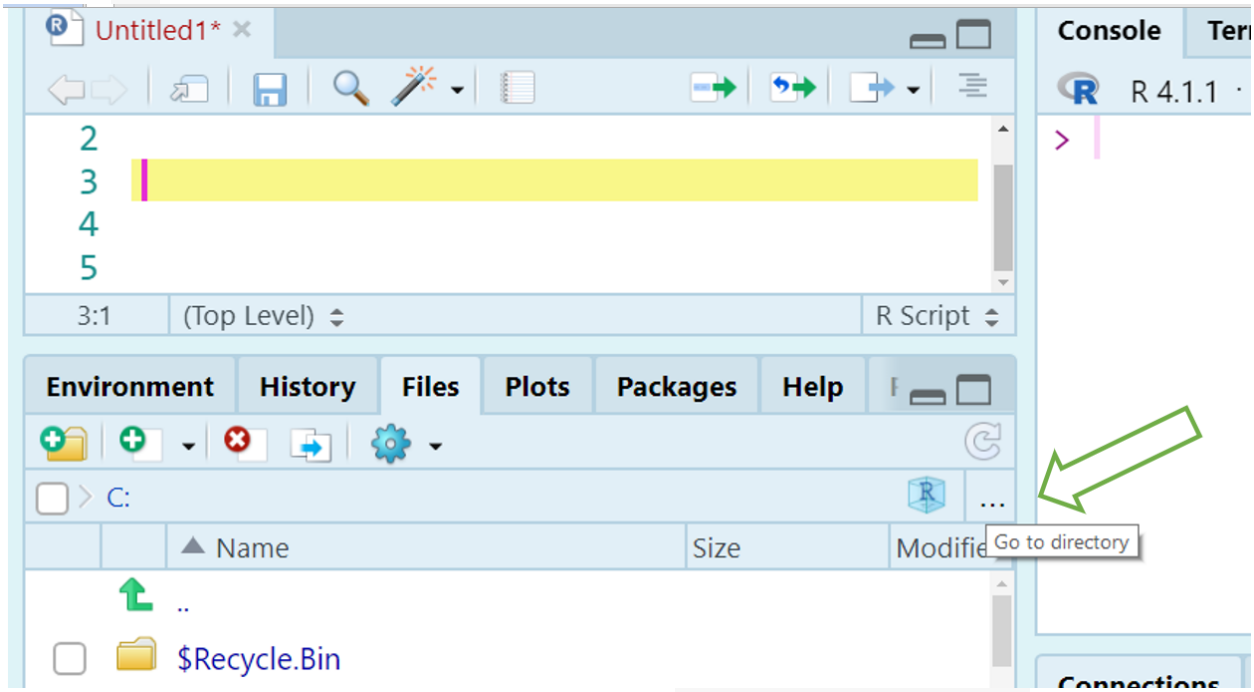
The first three files, `0_Custom_Path.R`, `1_Create_Folders.R` and `1_Install_Packages.R` can be run as a preliminary step to get ready to run the above main analysis files, as discussed in the following section.

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 a 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,

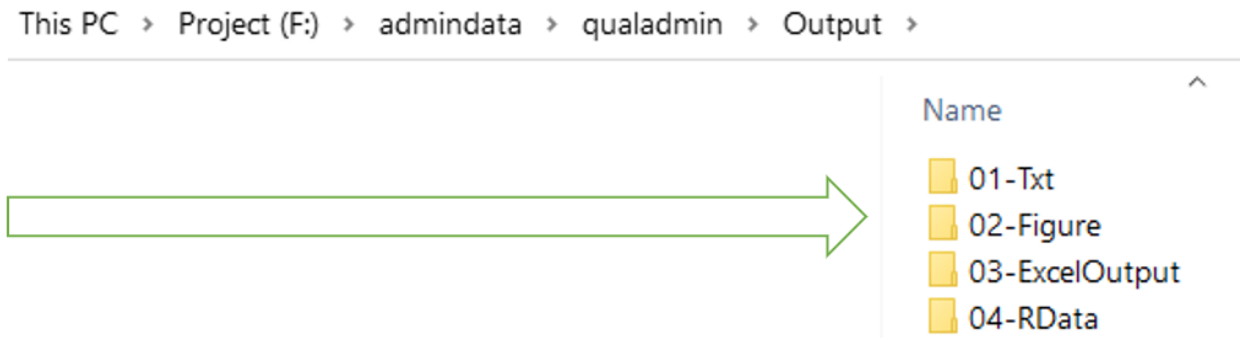
```
setwd("F:\admindata)`
Error: '\a' is an unrecognized escape in character string starting ""F:\a
"
```

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

3.2.3 Automatically create output folders

The main R script files produce a range of 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 the code 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. The installation should be performed once.

```
#---------------------------------
# Install packages (Run once)
#---------------------------------
install.packages("tidyverse")
install.packages("ggplot2")
install.packages("fastDummies")
install.packages("janitor")
```

Now, you're all set to proceed with quality measures indicators.

3.3 RUNNING Prep_auxiliary.R

This code file produces a frequency table for both weighted sample data and administrative data and calculates the distributions of each data source. From the frequency, we will prepare several ***auxiliary data objects*** to be used for R-indicator computation procedures. In essence, the auxiliary data file, pop_respmean is a summary of the distributions, containing the proportions of each category from the defined variables in row vectors. Please note that the proportions concerning the last categories are to be removed. The file then rearranges the frequency table and saves it in a format suitable for computing R indicators. This is called bivariate counts.

If the Census data or a weighted sample data are available, users consult 3.3.1 *Use the existing weighted sample data*. For a scenario where these data are unavailable, users can generate the data as shown in 3.3.2 *Generate a weighted sample data*. The overview is provided below.

```
Prep_auxiliary
 |
 ├─ A. Using an existing data
 |  ├─ (pop + admin) frequency table (for distance metrics)
 |  ├─ pop_respmean (for R indicators)
 |  └─ (pop) bivariate counts (for R indicators)
 |
 └─ B. Creating a weighted sample data
    ├─ (pop + admin) frequency table
    ├─ pop_respmean
    └─ (pop) bivariate counts
```

3.3.1 Use the existing weighted sample data

Open the `2_A_Prep_auxiliary.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")

# Disable scientific notation.
options(scipen = 999)

library("dplyr")     # data manipulation
library("ggplot2")   # visualisation
library("readr")     # read large csv file
library("readxl")    # read Excel file
```

```
    library("writexl")   # export to Excel
    library("janitor")   # cross-tabulation
```

The major steps are shown below.

- **Step 1**: read in the data.

```
    df  <- read_csv("custom_wtsample.csv")

    dim(df)       # obs = 1163659 (example)
    glimpse(df)   # Quick glance at the data
```

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

```
    names(df)     # variable names

    [1] "geog1"     "sex"       "agecode1"  "eth_code5"
    [5] "econg"
```

Users should maintain consistency by ensuring that both administrative data (denoted by `aa`) and the benchmark data(indicated by `df`) contain same variables of interest only. Rename variables to be consistent with the weighted sample data and sort as necessary. *Please note that sorting is not required.* The programme will detect the number of variables, and categories automatically. Notice the sample size for the example admin data is 1,033,664.

- **Step 2**: Obtain the frequency table of categorical variables (count of categories).

This procedure is to assess and calculate *the distribution* of categories in the benchmark and administrative data. Users can run the pre-defined function, `fn_freq_table()` to perform the task.

```
        fn_freq_table(df)  # benchmark data, df
        fn_freq_table(aa)  # admin data, aa
```

The function automatically obtains counts and proportions in oneway and two-way frequency tables before structuring the output in long form, organised by each variable, and by its discrete category. One can see the count, `n`, and the corresponding proportion, `p` by each variable in benchmark data. The corresponding counts and proportions for admin data are also displayed.

```
# one-way
   oneway_by        n      p v1    v2    by1      twby   by2 admin_n admin_p
 1 total      1163659 1      NA    NA    NA       total   NA 1033664  1
 2 geog1_1     116128 0.0998 1     1     geog1    oneway   0 137993   0.133
 3 geog1_2     150139 0.129  1     2     geog1    oneway   0 124051   0.120
 4 geog1_3     137520 0.118  1     3     geog1    oneway   0 131176   0.127
 5 geog1_4     170624 0.147  1     4     geog1    oneway   0 139867   0.135
 6 geog1_5      90873 0.0781 1     5     geog1    oneway   0 142304   0.138
 7 geog1_6     498375 0.428  1     6     geog1    oneway   0 358273   0.347
 8 sex_1       564905 0.485  2     1     sex      oneway   0 489228   0.473
 9 sex_2       598754 0.515  2     2     sex      oneway   0 544436   0.527
10 agecode1_1   82426 0.0708 3     1     agecode1 oneway   0  55659   0.0538
# two-way
```

```
   oneway_by      n       p v1    v2    by1   twby     by2 admin_n admin_p
 1 ""         56409 0.0485  1     1     geog1 sex        2    64231 0.0621
 2 ""         59719 0.0513  1     2     geog1 sex        2    73762 0.0714
 3 ""         72251 0.0621  2     1     geog1 sex        2    58021 0.0561
 4 ""         77888 0.0669  2     2     geog1 sex        2    66030 0.0639
 5 ""         65755 0.0565  3     1     geog1 sex        2    62060 0.0600
 6 ""         71765 0.0617  3     2     geog1 sex        2    69116 0.0669
 7 ""         85150 0.0732  4     1     geog1 sex        2    67305 0.0651
 8 ""         85474 0.0735  4     2     geog1 sex        2    72562 0.0702
 9 ""         44218 0.0380  5     1     geog1 sex        2    72237 0.0699
10 ""         46655 0.0401  5     2     geog1 sex        2    70067 0.0678
11 ""        241122 0.207   6     1     geog1 sex        2   165374 0.160
12 ""        257253 0.221   6     2     geog1 sex        2   192899 0.187
```

Note that **p** is obtained by dividing **n** by **popsize**. For example, the value of **p** for first category of **geog1** is calculated as **116128/1163659 = 0.0998**. One can carry out checks to see if the calculated proportions of a two-way table are accurate. The following code obtains the total observation size and confirms that the total proportion adds up to 1, for a chosen two-way table. Here, the total observation size can be viewed as the population size.

```
# Check the population size
        sum(pop_admin_freq_table[2:7, "n"])
## [1] 1163659

# Check whether the total adds up to 1
pop_admin_freq_table %>% filter(by2 == 2) %>%
 summarise(twoway_sum_p = sum(p) )
```

```
  twoway_sum_p
         <dbl>
1            1
```

- **Step 3**: Obtain pop respmean

Drawing upon the earlier frequency table, we only use information on oneway frequency tables to obtain pop_respmean row vectors in preparation for the computation of R-indicators.

We will treat the column, `p`, representing the distributions of benchmark data, as `popmeans`.

des_seq	oneway_by	popmean_c	n	p	admin_n	weighted_admin_n	respmean	rrate	finalwgt
1	total	popmean1	1163659	1	1033664	1163659	1	0.88829	1.12576
2	geog1_1	popmean2	116128	0.0998	137993	155347.1886	0.133498893	0.88829	1.12576
3	geog1_2	popmean3	150139	0.12902	124051	139651.8236	0.120010951	0.88829	1.12576
4	geog1_3	popmean4	137520	0.11818	131176	147672.8734	0.126903907	0.88829	1.12576
5	geog1_4	popmean5	170624	0.14663	139867	157456.8654	0.135311861	0.88829	1.12576
6	geog1_5	popmean6	90873	0.07809	142304	160200.3459	0.137669494	0.88829	1.12576
-999	geog1_6		498375	0.42828	358273	403329.9031	0.346604893	0.88829	1.12576
7	sex_1	popmean7	564905	0.48546	489228	550753.9832	0.473294997	0.88829	1.12576
-999	sex_2		598754	0.51454	544436	612905.0168	0.526705003	0.88829	1.12576
8	agecode1_1	popmean8	82426	0.07083	55659	62658.75205	0.053846318	0.88829	1.12576
9	agecode1_2	popmean9	94643	0.08133	64496	72607.10527	0.062395517	0.88829	1.12576
10	agecode1_3	popmean10	110296	0.09478	81222	91436.58993	0.078576791	0.88829	1.12576
11	agecode1_4	popmean11	120398	0.10347	113740	128044.0981	0.110035756	0.88829	1.12576
12	agecode1_5	popmean12	119393	0.1026	112863	127056.8054	0.109187318	0.88829	1.12576
13	agecode1_6	popmean13	101711	0.08741	96444	108572.9295	0.093303046	0.88829	1.12576
14	agecode1_7	popmean14	94209	0.08096	89764	101052.8436	0.086840598	0.88829	1.12576
15	agecode1_8	popmean15	100159	0.08607	95381	107376.2452	0.092274666	0.88829	1.12576
16	agecode1_9	popmean16	77799	0.06686	73902	83196.01671	0.071495186	0.88829	1.12576
17	agecode1_10	popmean17	65833	0.05657	62717	70604.3758	0.060674455	0.88829	1.12576
18	agecode1_11	popmean18	57305	0.04925	54730	61612.91974	0.052947573	0.88829	1.12576
19	agecode1_12	popmean19	51263	0.04405	48803	54940.5321	0.047213601	0.88829	1.12576
20	agecode1_13	popmean20	43678	0.03754	41664	46903.72169	0.040307102	0.88829	1.12576
-999	agecode1_14		44546	0.03828	42279	47596.06493	0.040902073	0.88829	1.12576
21	eth_code5_1	popmean21	1081812	0.92966	962692	1083761.464	0.931339391	0.88829	1.12576
22	eth_code5_2	popmean22	10487	0.00901	8928	10050.7975	0.008637236	0.88829	1.12576
23	eth_code5_3	popmean23	46446	0.03991	40147	45195.94169	0.038839507	0.88829	1.12576
24	eth_code5_4	popmean24	16268	0.01398	14355	16160.30446	0.013887491	0.88829	1.12576
-999	eth_code5_5		8646	0.00743	7542	8490.492247	0.007296375	0.88829	1.12576
25	econg_1	popmean25	689140	0.59222	609471	686118.9075	0.589621966	0.88829	1.12576
26	econg_2	popmean26	27744	0.02384	23289	26217.85653	0.022530532	0.88829	1.12576
-999	econg_3		446775	0.38394	400904	451322.236	0.387847502	0.88829	1.12576

Figure 1 Illustration of popmean and respmean

In terms of administrative data, we obtain the proportions based on weighted sample counts. The weight, indicated by `finalwgt`, is derived by the inverse of *response rate* (finalwgt = 1/rrate). To calculate the *response rate*, we divide the total admin sample size by the population size. In R, we can express `rrate` as `nrow(aa)/popsize`, or `1033664/1163659=0.88829`. By taking the inverse of response rate, we compute `finalwgt` of 1.12576.

Now, we apply the weight, 1.12576 (in this example), to the *raw* admin count of each category in order to obtain `weighted_admin_n`. The `respmeans` are the ratio of the weighted sample counts to the population size. For example, respmean for the first dummy column, *geog1_1*, is computed by `(137993 * finalwgt)/popsize = weighted_admin_n/popsize = 0.1334`.

These two *popmeans* and *respmeans* columns will be transposed before we save it as row vectors. Note that we need to remove the **last** category for each categorical variable. In this example, from 5 variables, geog1 (6), sex (2), agecode1 (14) and eth_cod5 (5) and econg (3), we now have nvar-1 for each variable. We also add another row indicating the total observation size (1+5+1+13+4+3=26). As such, we keep 26 `popmeans` and `respmeans`.

The resulting `pop_respmean` (stored in row vectors) is printed below.

```
  popmean1   popmean2  popmean3 popmean4  popmean5   popmean6  popmean7
         1 0.09979556 0.1290232 0.118179 0.1466272 0.07809247 0.4854558
    popmean8   popmean9  popmean10 popmean11 popmean12  popmean13  popmean14
 0.07083347 0.08133225 0.09478378  0.103465 0.1026014 0.08740619 0.08095928
   popmean15  popmean16  popmean17  popmean18  popmean19  popmean20 popmean21
 0.08607247 0.06685722 0.05657413 0.04924553 0.04405328 0.03753505 0.9296641
   popmean22  popmean23  popmean24 popmean25  popmean26 respmean1 respmean2
 0.00901209 0.03991375 0.01398004 0.5922182 0.02384204          1 0.1334989
  respmean3 respmean4 respmean5 respmean6 respmean7  respmean8  respmean9
0.120011 0.1269039 0.1353119 0.1376695  0.473295 0.05384632 0.06239552
respmean10 respmean11 respmean12 respmean13 respmean14 respmean15 respmean16
0.07857679  0.1100358  0.1091873 0.09330305  0.0868406 0.09227467 0.07149519
respmean17 respmean18 respmean19 respmean20 respmean21  respmean22 respmean23
0.06067446 0.05294757  0.0472136  0.0403071  0.9313394 0.008637236 0.03883951
respmean24 respmean25 respmean26
0.01388749   0.589622 0.02253053
```

- **Step 4**: Obtain bivariate counts of benchmark data

Another intermediate data object we need in the calculation of R-indicators are the matrix containing bivariate counts in a benchmark data. The procedures are largely similar to those seen in the frequency table. Essentially, both computations are based on frequency tables. A notable difference is that we store the information on bivariate counts in a form of matrix. Another important distinction is that the frequency table we prepared earlier removes duplicate entries.

```
# Use benchmark data (set as pop); dim(df)
  pop <- df
fn_bivariate_counts_popcov(pop)
  head(popcov)
           c1     c2     c3     c4     c5    c6     c7    c8    c9    c10
  [1,] 1163659 116128 150139 137520 170624 90873 564905 82426 94643 110296
  [2,]  116128 116128      0      0      0     0  56409  8073 11333  12864
  [3,]  150139      0 150139      0      0     0  72251  9408 14292  19338
  [4,]  137520      0      0 137520      0     0  65755  7390  9596  14194
  [5,]  170624      0      0      0 170624     0  85150 16707 22627  17094
  [6,]   90873      0      0      0      0 90873  44218  6748  5890   7526

  c11    c12    c13   c14    c15   c16   c17   c18   c19   c20     c21
  [1,]120398 119393 101711 94209 100159 77799 65833 57305 51263 43678 1081812
  [2,] 12883  12351  10072  9232   9518  6655  5363  4735  4346  4065   99895
  [3,] 18534  16473  12719 11327  10889  7891  7120  6355  5594  4872  115591
  [4,] 17079  16106  12803 11535  12102  8600  6594  5719  5202  4925  126584
  [5,] 15426  14635  12908 11809  12066  9932  8377  7871  7447  6733  159404
  [6,]  8876   9585   8768  7984   8313  6336  5397  4612  4114  3437   88809

   c22   c23   c24    c25   c26
  [1,] 10487 46446 16268 689140 27744
```

```
 [2,]  1744 10135  1678  70776  2689
 [3,]  3099 18156 10638  91611  4527
 [4,]  1931  6158  1164  86527  3270
 [5,]  1622  7033  1738  90768  4600
 [6,]   409  1349   182  58258  1343
```

3.3.2 Generate a weighted sample data

- **Step 1**: Generate a weighted sample data

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`.

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)

Variable	Category	Description	N	(%)
	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.

Then, we draw a random sample 1:50.

- **Step 2**: Obtain the frequency table of categorical variables (count of categories).

From the randomly selected sample (`1163659/50 = 23273`), we then obtain frequency table of categorical variables (count of categories). As seen in 3.3.1, users can run the pre-defined function, `fn_freq_table(df)` 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.

```
fn_freq_table(df)
```

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). The remaining steps are consistent with those illustrated in the section 3.3.1.

3.4 RUNNING 3_Distance_Metrics.R

To calculate distance metrics, we need one-way, and two-way frequency tables of categorical variables, and proportions of each sub-category by each data source. Previously, we dealt with frequency tables for both benchmark and administrative data already. Here, 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 provide standardised metrics.

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

Step 1: Prepare an auxiliary data file

The first step is to ensure we have auxiliary information. Simply *source* one of the previous files as below to update it.

```
        # source("2_A_Prep_auxiliary.R")
        source("2_B_Prep_auxiliary_CreateRandomSample.R")
```

Step 2: Compute distance metrics

Run the functions to compute three types of distance metrics.

```
        fn_distance_metrics()
```

Step 3: Reshape for plotting

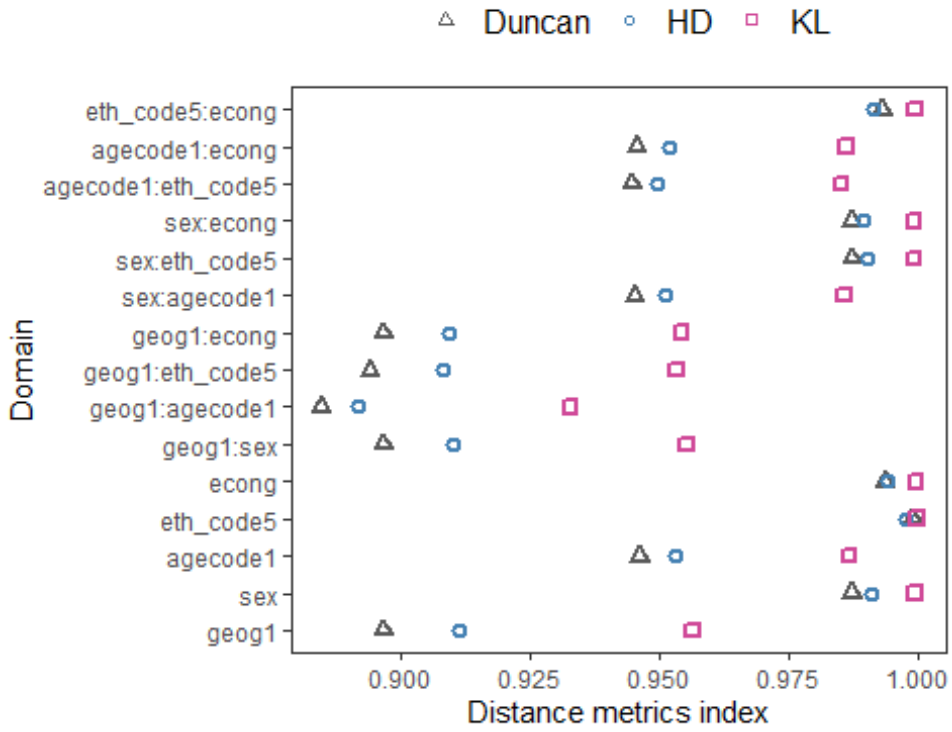
From wide form, the outputs have been reshaped to long form. Then, we keep standardised solutions. The results are as follows:

```
      fn_distance_metrics_long()
```

```
## # A tibble: 9 x 5
##   domain_id domain   indicator  index           ref
##       <int> <chr>    <chr>      <dbl>          <fct>
## 1         1 geog1    Std_Duncan 0.897           1
## 2         1 geog1    Std_HD     0.911           2
## 3         1 geog1    Std_KL     0.957           3
## 4         2 sex      Std_Duncan 0.987           1
## 5         2 sex      Std_HD     0.991           2
## 6         2 sex      Std_KL     1.00            3
## 7         3 agecode1 Std_Duncan 0.946           1
## 8         3 agecode1 Std_HD     0.953           2
## 9         3 agecode1 Std_KL     0.987           3
```

Step 4: Visualise the distance metrics

```
        plot(p)
```



3.5 RUNNING 3B_R-indicator.R

This procedure computes the overall R-indicator and partial indicators when the underlying administrative data is individual-level microdata. For very large datasets, it is possible to calculate the R-indicator and partial indicators from a table aggregated to the level of the covariates (variables of interest) in the model. This procedure is described in Section 3.6.

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/1_Functions.R` file for more details on the algorithm and operationalisation.

Step 1: Prepare an Auxiliary data file

The first step is to ensure we have auxiliary information. Simply *source* one of the previous files as below to update it.

```
        # source("2_A_Prep_auxiliary.R")
        source("2_B_Prep_auxiliary_CreateRandomSample.R")
```

Step 2: Read in the administrative data

Read in the administrative data and examine the data.

```
aa  <- read_csv("public_release_admin.csv")

nrow(aa) # 1033664
## [1] 1033664

# rename, keep variables of interest and sort
aa <- aa %>%
 dplyr::select(-person_id) %>%
 arrange(geog1, sex, agecode1, eth_code5, econg)

head(aa)
##   geog1  sex agecode1 eth_code5 econg
## 1 <dbl> <dbl> <dbl>    <dbl>   <dbl>
## 2    1   1        1         1     1
## 3    1   1        1         1     1
## 4    1   1        1         1     1
## 5    1   1        1         1     1
## 6    1   1        1         1     1
```

Step 3: Compute R-indicator

Once steps 1 and 2 are completed, we are set to carry out computing the R-indicator. These procedures are automated via pre-defined functions using complex matrix and data management syntax.

By running the single function below, users can produce estimates of overall and partial R indicators. Please note that some procedures can be computationally intensive.

```
 fn_RUN_R_indicator()
```

Once the function is executed, users can proceed with the next steps 4 and 5 to export the output.

The function, `fn_RUN_R_indicator()`, consists of multiple sub-functions. Note that we have several sub-functions that permit us to compute overall R-indicators, along with additional functions for obtaining partial R-indicators.

```
fn_design_matrix()
fn_des_pop_respmean()
fn_gh()
fn_R_indicator()
fn_rindicatorall()
fn_partial()
```

Each function will perform a particular task and generate intermediate properties stored in objects including `design_matrix`, `des_pop_respmean`, `gh`, `R_indicator` and , `rindicatorall` as indicated in the function name. The final output, partial R-indicators, can be found in the data object, `partial`.

Let's go over one function at a time.

fn_design_matrix()

The utility of `fn_design_matrix()` 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 `design_matrix` object, which contains the original variables, and the design matrix with weights. The last few columns are printed below. As des1 denotes intercept, all values of des1 is set at 1. As the first six observations show geog1 = 1, the dummy column *geog1_1* is initially created internally (not shown) before being renamed as `des2`. Then, the next gego1 dummy column, *geog1_2* (where, geog1 = 2) is generated, which will subsequently be named as `des3`. In the case of observation 1, since the first six rows are geog1 = 1, the values of columns from `des3` to `des6` (geog1 = 2, geog1 = 3, geog1 = 4, geog1 = 5, respectively) become 0. We also notice that the dummy for the last geog1 category, *geog1_6* (geog1 = 6), is omitted. As such, the next column, `des7` concerns *sex_1* (where sex = 1) while `des8` denotes *agecode1_1* (agecode1=1).

```
head(design_matrix)
```

```
##   geog1 sex agecode1 eth_code5 econg des1 des2 des3 des4 des5 des6 des7 des8
## 1     1   1        1         1     1    1    1    0    0    0    0    1    1
## 2     1   1        1         1     1    1    1    0    0    0    0    1    1
## 3     1   1        1         1     1    1    1    0    0    0    0    1    1
## 4     1   1        1         1     1    1    1    0    0    0    0    1    1
## 5     1   1        1         1     1    1    1    0    0    0    0    1    1
## 6     1   1        1         1     1    1    1    0    0    0    0    1    1
##   des9 des10 des11 des12 des13 des14 des15 des16 des17 des18 des19 des20 des21
## 1    0     0     0     0     0     0     0     0     0     0     0     0     1
```

```
## 2    0     0     0     0     0     0     0     0     0     0     0     0     1
## 3    0     0     0     0     0     0     0     0     0     0     0     0     1
## 4    0     0     0     0     0     0     0     0     0     0     0     0     1
## 5    0     0     0     0     0     0     0     0     0     0     0     0     1
## 6    0     0     0     0     0     0     0     0     0     0     0     0     1
##   des22 des23 des24 des25 des26 finalwgt piinv
## 1     0     0     0     1     0 1.125753     1
## 2     0     0     0     1     0 1.125753     1
## 3     0     0     0     1     0 1.125753     1
## 4     0     0     0     1     0 1.125753     1
## 5     0     0     0     1     0 1.125753     1
## 6     0     0     0     1     0 1.125753     1
```

Recall that the weights are calculated by the inverse of *rrate* (`finalwgt = 1/rrate`).

fn_des_pop_respmean()

This function allows us to combine design matrix, popmean and respmean vectors in a single dataset, `des_pop_respmean`.

```
   head(des_pop_respmean, n = 3)
```

```
##   geog1 sex agecode1 eth_code5 econg des1 des2 des3 des4 des5 des6 des7 des8
## 1     1   1        1         1     1    1    1    0    0    0    0    1    1
## 2     1   1        1         1     1    1    1    0    0    0    0    1    1
## 3     1   1        1         1     1    1    1    0    0    0    0    1    1
##   des9 des10 des11 des12 des13 des14 des15 des16 des17 des18 des19 des20 des21
## 1    0     0     0     0     0     0     0     0     0     0     0     0     1
## 2    0     0     0     0     0     0     0     0     0     0     0     0     1
## 3    0     0     0     0     0     0     0     0     0     0     0     0     1
##   des22 des23 des24 des25 des26 finalwgt piinv popmean1   popmean2  popmean3
## 1     0     0     0     1     0 1.125753     1        1 0.09732308 0.1275298
## 2     0     0     0     1     0 1.125753     1        1 0.09732308 0.1275298
## 3     0     0     0     1     0 1.125753     1        1 0.09732308 0.1275298
##    popmean4  popmean5   popmean6 popmean7   popmean8   popmean9  popmean10
## 1 0.1181197 0.1504748 0.07931938 0.485842 0.07270227 0.08017875 0.09556138
## 2 0.1181197 0.1504748 0.07931938 0.485842 0.07270227 0.08017875 0.09556138
## 3 0.1181197 0.1504748 0.07931938 0.485842 0.07270227 0.08017875 0.09556138
##   popmean11 popmean12  popmean13  popmean14  popmean15  popmean16  popmean17
## 1 0.1067761 0.1015769 0.08589352 0.08232716 0.08215529 0.06741718 0.05809307
## 2 0.1067761 0.1015769 0.08589352 0.08232716 0.08215529 0.06741718 0.05809307
## 3 0.1067761 0.1015769 0.08589352 0.08232716 0.08215529 0.06741718 0.05809307
##   popmean18  popmean19  popmean20 popmean21   popmean22  popmean23  popmean24
## 1 0.0498002 0.04352683 0.03630817 0.9309071 0.008980364 0.03918704 0.01400765
## 2 0.0498002 0.04352683 0.03630817 0.9309071 0.008980364 0.03918704 0.01400765
## 3 0.0498002 0.04352683 0.03630817 0.9309071 0.008980364 0.03918704 0.01400765
##   popmean25  popmean26 respmean1 respmean2 respmean3 respmean4 respmean5
## 1  0.594079 0.02427706         1  0.133498 0.1200103 0.1269033 0.1353113
## 2  0.594079 0.02427706         1  0.133498 0.1200103 0.1269033 0.1353113
## 3  0.594079 0.02427706         1  0.133498 0.1200103 0.1269033 0.1353113
##   respmean6 respmean7  respmean8  respmean9 respmean10 respmean11 respmean12
## 1 0.1376694 0.4732944 0.05384609 0.06239505 0.07857603  0.1100357  0.1091866
```

```
## 2 0.1376694 0.4732944 0.05384609 0.06239505 0.07857603  0.1100357  0.1091866
## 3 0.1376694 0.4732944 0.05384609 0.06239505 0.07857603  0.1100357  0.1091866
##   respmean13 respmean14 respmean15 respmean16 respmean17 respmean18 respmean19
## 1 0.09330297 0.08684054 0.09227431 0.07149487 0.06067374 0.05294719 0.04721351
## 2 0.09330297 0.08684054 0.09227431 0.07149487 0.06067374 0.05294719 0.04721351
## 3 0.09330297 0.08684054 0.09227431 0.07149487 0.06067374 0.05294719 0.04721351
##   respmean20 respmean21  respmean22 respmean23 respmean24 respmean25 respmean26
## 1 0.04030679  0.9313393 0.008636618   0.038839 0.01388734  0.5896214 0.02252997
## 2 0.04030679  0.9313393 0.008636618   0.038839 0.01388734  0.5896214 0.02252997
## 3 0.04030679  0.9313393 0.008636618   0.038839 0.01388734  0.5896214 0.02252997
##   seq responsesamp1
## 1   1             1
## 2   2             1
## 3   3             1
```

fn_gh()

Once the combined dataset, `des_pop_respmean`, is obtained, we compute the difference from the mean vectors and the weight variables indicated by des1-des26 columns. The differences between des and *popmean* columns are stored in `psam`. Similarly, `rsam` columns will be created, representing the differences between des and *respmeans*. We then name the dataset as gh.

						total						des - popmean		
seq	geog1	sex	agecode1	eth_code5	econg	des1	des2	des3	popmean1	popmean2	popmean3	psam1	psam2	psam3
137989	1	2	14	4	3	1	1	0	1	0.09732	0.12753	0	0.90268	-0.12753
137990	1	2	14	5	3	1	1	0	1	0.09732	0.12753	0	0.90268	-0.12753
137991	1	2	14	5	3	1	1	0	1	0.09732	0.12753	0	0.90268	-0.12753
137992	1	2	14	5	3	1	1	0	1	0.09732	0.12753	0	0.90268	-0.12753
137993	1	2	14	5	3	1	1	0	1	0.09732	0.12753	0	0.90268	-0.12753
137994	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247
137995	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247
137996	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247
137997	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247
137998	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247
137999	2	1	1	1	1	1	0	1	1	0.09732	0.12753	0	-0.09732	0.87247

						total						des - respmean		
seq	geog1	sex	agecode1	eth_code5	econg	des1	des2	des3	respmean1	respmean2	respmean3	rsam1	rsam2	rsam3
137989	1	2	14	4	3	1	1	0	1	0.13350	0.12001	0	0.86650	-0.12001
137990	1	2	14	5	3	1	1	0	1	0.13350	0.12001	0	0.86650	-0.12001
137991	1	2	14	5	3	1	1	0	1	0.13350	0.12001	0	0.86650	-0.12001
137992	1	2	14	5	3	1	1	0	1	0.13350	0.12001	0	0.86650	-0.12001
137993	1	2	14	5	3	1	1	0	1	0.13350	0.12001	0	0.86650	-0.12001
137994	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999
137995	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999
137996	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999
137997	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999
137998	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999
137999	2	1	1	1	1	1	0	1	1	0.13350	0.12001	0	-0.13350	0.87999

Figure 2 Illustration of psam and rsam columns

Excerpts of the code from the function is shown below. This concludes the pre-matrix preparation part.

```
  df <-  data.frame(des_pop_respmean)
  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
  rpsam        <-  data.frame(rsam, psam)
  colnames(rpsam) <- c(paste0("rsam", 1:numcat),
                       paste0("psam", 1:numcat))
  gh <-  cbind(des_pop_respmean, rpsam)
```

fn_R_indicator()

The `fn_R_indicator()` 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 this manual, we demonstrate using *prop_mix*. From version v3.0 onwards, the auxiliary data, `popcov`, created previously in section 3.3 is entered as part of procedures obtaining propensity scores.

R-indicator is computed by 1 - standard deviation of propensity scores, or 1-sqrt(variance). In this example, the calculation would be 1-sqrt(0.06349736) as the variance of propensity scores is estimated as 0.06349736.

```
   print(R_indicator)
```

```
## [1] 0.7480132
```

fn_rindicatorall()

In this step, we take the sum of the weights, finalwgt, and the average of the propensity scores for each of the categories of variables, which are stored in fbar and mrphat columns.

```
   print(rindicatorall)
```

```
##       fbar1    fbar2     fbar3       fbar4     fbar5   mrphat1   mrphat2
## 1  155346.0 550749.7  62658.27 1083753.082 686113.60 1.2004888 0.9362283
## 2  139650.7 612900.3  72606.54   10050.720  26217.65 0.9041243 0.9809375
## 3  147671.7       NA  91435.88   45195.592 451318.75 1.0212640        NA
## 4  157455.6       NA 128043.11   16160.179        NA 0.8602353        NA
## 5  160199.1       NA 127055.82    8490.427        NA 1.3362734        NA
## 6  403326.8       NA 108572.09          NA        NA 0.7531388        NA
```

```
## 7        NA       NA 101052.06          NA         NA         NA         NA
## 8        NA       NA 107375.41          NA         NA         NA         NA
## 9        NA       NA  83195.37          NA         NA         NA         NA
## 10       NA       NA  70603.83          NA         NA         NA         NA
## 11       NA       NA  61612.44          NA         NA         NA         NA
## 12       NA       NA  54940.11          NA         NA         NA         NA
## 13       NA       NA  46903.36          NA         NA         NA         NA
## 14       NA       NA  47595.70          NA         NA         NA         NA
##      mrphat3   mrphat4   mrphat5 seq
## 1  0.6487130 0.9601892 0.9530622   1
## 2  0.7066057 0.9244880 0.8909178   2
## 3  0.7677695 0.9518284 0.9739849   3
## 4  0.9860914 0.9520912        NA   4
## 5  1.0216912 1.0058556        NA   5
## 6  1.0303194        NA        NA   6
## 7  1.0059449        NA        NA   7
## 8  1.0571924        NA        NA   8
## 9  1.0104442        NA        NA   9
## 10 0.9975692        NA        NA  10
## 11 1.0125800        NA        NA  11
## 12 1.0291411        NA        NA  12
## 13 1.0479060        NA        NA  13
## 14 1.0296841        NA        NA  14
```

The function also computes the overall mean of propensity scores.

```
mrphatall <- mean(gh_prop_mix$roi)
mrphatall
## [1] 0.9597769
```

fn_partial()

The category and variable-level partical R-indicators can be otained by running `fn_partial()`. The output below shows variable-level and category-level R-indicator estimates.

```
    partial[1:17, c(1:4, 6:8)]

##    seq level    domain R_indicator_summary  count n_cat domain_n
## 1    1     0   Overall         0.748013157     NA  <NA>       NA
## 2    2     0 mrphatall         0.959776861     NA  <NA>       NA
## 3    3    99   resppop  1033664.0000000000     NA  <NA>       NA
## 4    4     1     geog1         0.210337414     NA  <NA>       NA
## 5    5     1       sex         0.022322665     NA  <NA>       NA
## 6    6     1  agecode1         0.123628959     NA  <NA>       NA
## 7    7     1 eth_code5         0.005447982     NA  <NA>       NA
## 8    8     1     econg         0.014550224     NA  <NA>       NA
## 9    9    99      des1                  NA      0     1        0
## 10  10     3   geog1_1         0.087950135 137993     1        1
## 11  11     3   geog1_2        -0.019279482 124051     2        1
## 12  12     3   geog1_3         0.021903908 131176     3        1
## 13  13     3   geog1_4        -0.036616129 139867     4        1
```

```
## 14  14     3   geog1_5          0.139694673 142304     5        1
## 15  15     3   geog1_6         -0.121654343 358273     6        1
## 16  16     3     sex_1         -0.016200550 489228     1        2
## 17  17     3     sex_2          0.015357199 544436     2        2
```

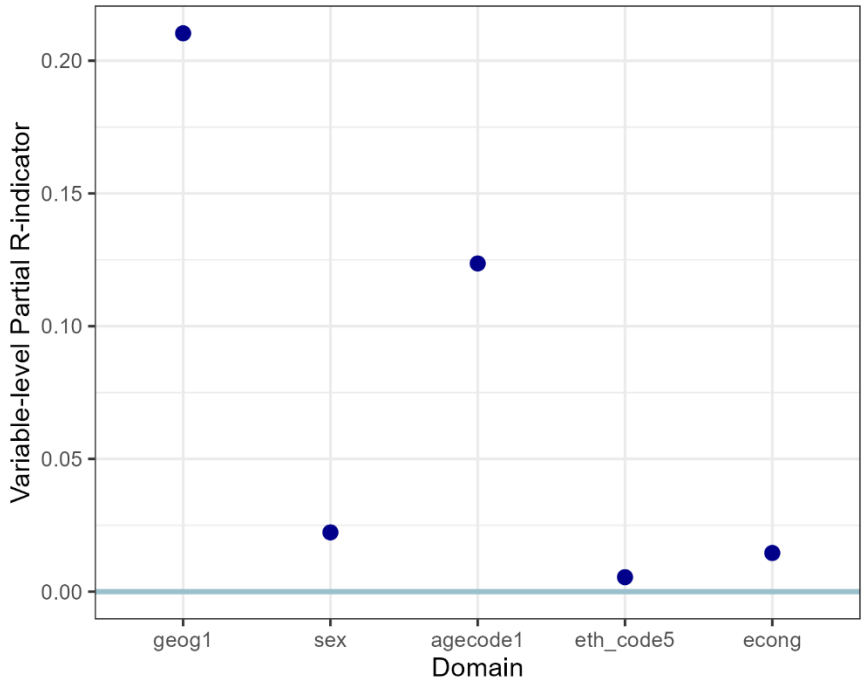
Step 4: Save in Excel

At this stage, users can inspect the output accordingly. Here, we can see the overall R-indicator is estimated as 0.748 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 (0.21) compared to *econg* (0.014).

Step 5: Visualise using scatterplots

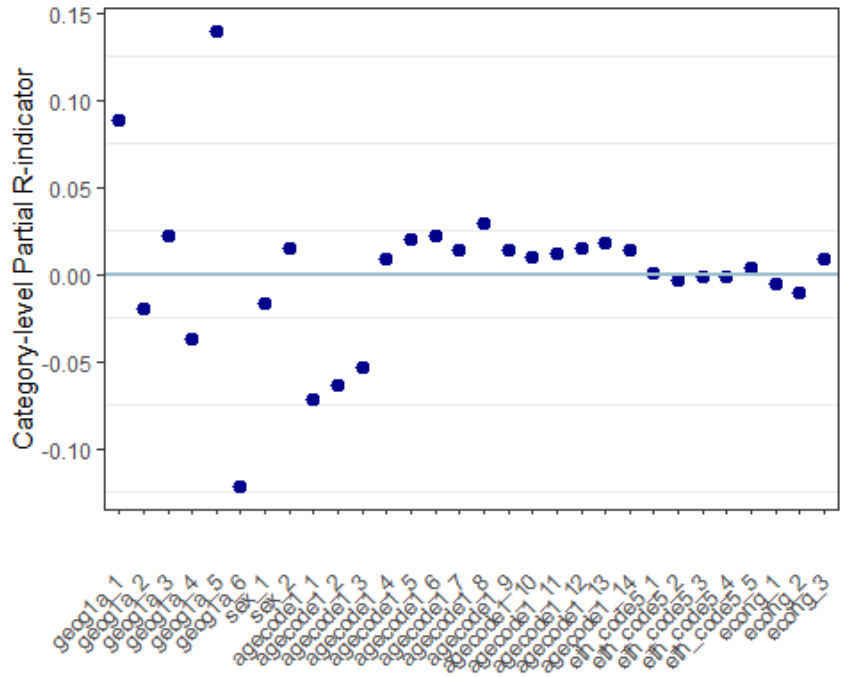
The visualisation of R-indicator by the variable level is shown as an example:

```
plot(p1)
```



And for R-indicator by the category-level:

```
plot(p2)
```



3.6 RUNNING 3B_Table-based R-indicator

For very large administrative datasets, it is possible to aggregate the microdata to a frequency table spanned by the variables of interest (covariates) in the model prior to calculating R-indicators and partial indicators. The computation of the indicators will then be based on the *frequencies* in the table rather than the microdata. The procedures are less computationally intensive than those of microdata level R-indicators described in Section 3.5.

Follow the instructions given for Section 3.5 Step 1 and 2 and proceed with the following steps.

Step 3: Compute R-indicator

In the table-based R-indicator approach, administrative data will be reduced in the form of a table defined by the covariates of interest. The principle is largely the same as the earlier approach in 3.5, but the dataset have records of aggregates defined by the cells of the table (spanned by the variables of interest), and we need to use the aggregated sums as weights at the cell level: the variable 'piinv' is now the cell count in the table and the variable 'finalwgt' is the aggregated inverse response rate (defined as the population size divided by the administrative size) in the cell of the table.

The overall and partial R indicators can be computed by running the single function below.

```
fn_RUN_table_based_R_indicator()
```

After the function is executed, users can proceed with the next steps by producing a summary table in an Excel file and visualizing the variable-level and category-level R-indicators.

The function, `fn_RUN_table_based_R_indicator()`, is composed of six sub-functions:

```
fn_t_design_matrix()
fn_des_pop_respmean()
fn_gh()
fn_t_R_indicator()
fn_t_rindicatorall()
fn_partial()
```

Some functions can be used for both microdataset-based and table-based R-indicator computation. The function names containing `t` indicates that the computation concerns table-based R indicators. Users can run each sub-function to understand the procedure as necessary.

fn_t_design_matrix()

By running `fn_t_design_matrix()`, users can build a table spanned by the variables that will define the pop-based R-indicator, merge with the administrative data, identify missing cells, and perform a data reduction to organise the data characterised by the unique pattern, raw count, and aggregate weights for each pattern.

Build a Table and identify missing cells

In the provided dataset, we have 5 variables: geog1 (6), sex (2), agecode1 (14) and eth_cod5 (5) and econg (3). As such, the total theoretically-possible combinations of each variable will be 2520(6×2×1×4×5×3). We can verify that the size of 2520 as the last row number indicates 2520.

```
nrow(pp)

## [1] 2520
```

Let's request R to print the first 30 rows of the untreated table, `pp`, that illustrates the structure of the table.

```
head(pp, n = 30)

##    pp_seq geog1 sex agecode1 eth_code5 econg
## 1       1     1   1        1         1     1
## 2       2     1   1        1         1     2
## 3       3     1   1        1         1     3
## 4       4     1   1        1         2     1
## 5       5     1   1        1         2     2
## 6       6     1   1        1         2     3
## 7       7     1   1        1         3     1
## 8       8     1   1        1         3     2
## 9       9     1   1        1         3     3
## 10     10     1   1        1         4     1
## 11     11     1   1        1         4     2
## 12     12     1   1        1         4     3
## 13     13     1   1        1         5     1
## 14     14     1   1        1         5     2
## 15     15     1   1        1         5     3
## 16     16     1   1        2         1     1
## 17     17     1   1        2         1     2
## 18     18     1   1        2         1     3
## 19     19     1   1        2         2     1
## 20     20     1   1        2         2     2
## 21     21     1   1        2         2     3
## 22     22     1   1        2         3     1
## 23     23     1   1        2         3     2
## 24     24     1   1        2         3     3
## 25     25     1   1        2         4     1
## 26     26     1   1        2         4     2
## 27     27     1   1        2         4     3
## 28     28     1   1        2         5     1
## 29     29     1   1        2         5     2
## 30     30     1   1        2         5     3
```

The unique combination of each variable category is indicated by the pattern identifier, `pp_seq`. As seen, pp_seq spans up to 2520. Our goal is to examine the frequency of each pattern using the administrative data, before computing weights. This is carried out in several stages as depicted in Figure 1 below.

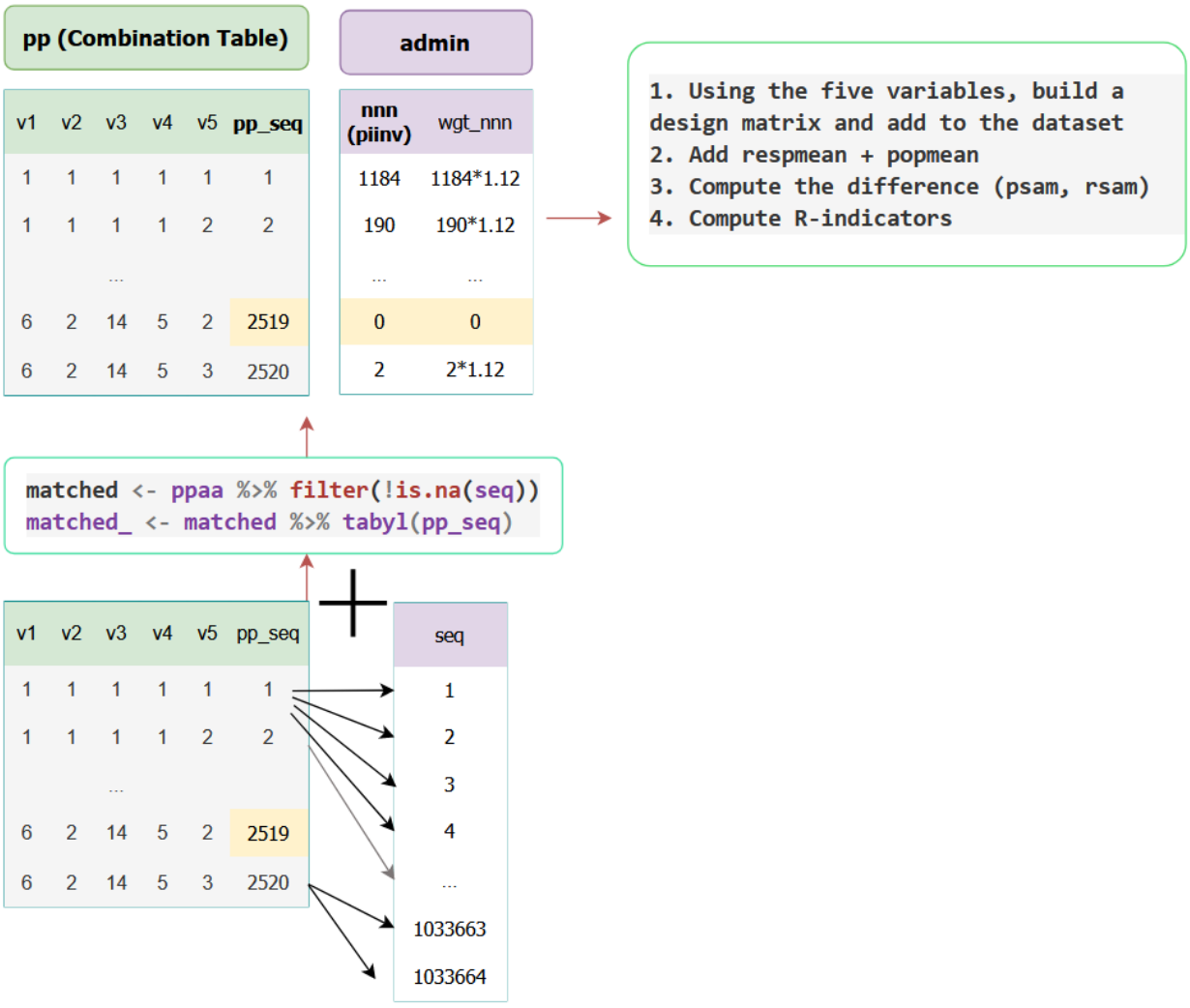


Figure 3 Illustration of obtaining pp combination table and the frequency of each combination

By merging this table with the administrative data, we can identify the *pp_seq* by each row/observation in the administrative data. As this is theoretical, not all this unique combination will be found in the administrative data, and unfound combination patterns or missing cells are to be expected. We will examine those missing cells shortly.

```
## >> Merge ----
# Recall  nrow(aa) = 1033664
ppaa  <- aa %>% dplyr::full_join(pp)
ppaa  <- ppaa %>% arrange(seq)
  nrow(ppaa)             # 1034120
```

Once we merge the `pp` table, consisting of 2520 rows, with the admin data, `aa`, the identifier of the pp, `pp_seq`, is distributed across the admin data. If the unique combination pattern

between two data sources is matched, the observation in the admin data, `seq` will display a corresponding pp_seq. If the matching pattern is not found in admin data, `seq` will indicate missing. We can verify the extent of missing cells (n = 456).

The following output displays the last 11 rows. As seen, the pattern identifier 116, which corresponds to the theoretically possible combination pattern of 1-1-8-4-2 among geog1, sex, agecode1, eth_code5, and econg variables' categories, is missing in the admin data.

```
missing  <- nrow(mi); missing

## [1] 456

ppaafrom <- nrow(ppaa) - missing - 4
ppaato   <- ppaafrom + 10
print(ppaa[ppaafrom: ppaato, ])

## # A tibble: 11 × 8
##    geog1 sex   agecode1 eth_code5 econg     seq finalwgt pp_seq
##    <fct> <fct> <fct>    <fct>     <fct>   <int>    <dbl>  <int>
##  1 6     2     14       4         3     1033660     1.13   2517
##  2 6     2     14       4         3     1033661     1.13   2517
##  3 6     2     14       4         3     1033662     1.13   2517
##  4 6     2     14       5         3     1033663     1.13   2520
##  5 6     2     14       5         3     1033664     1.13   2520
##  6 1     1     8        4         2          NA    NA       116
##  7 1     1     11       2         2          NA    NA       155
##  8 1     1     11       4         2          NA    NA       161
##  9 1     1     11       5         2          NA    NA       164
## 10 1     1     12       2         2          NA    NA       170
## 11 1     1     12       3         2          NA    NA       173
```

Organise the data by the Table and add weights

Please note that the weights, `finalwgt` column was computed in the administrative data. To obtain finalwgt, We first calculate the *response rate*, which is the admin size divided by the population size, `nrow(aa)/popsize`. The weights are computed by the inverse of *response rate* (finalwgt = 1/rrate). This can be treated as the *global weight*, as all observations have the consistent value, 1.125753.

At this stage, we reduce the data and organise by `pp_seq` (n = 2520), the combination pattern, and the raw count, `nnn`. We then apply appropriate weights, depending on the missingness represented by seq. For the non-missing seq, we tabulate pp_seq to obtain the raw count of each entry before applying the global weight, 1.125753 to the raw count. See the `wgt_nnn` column (wgt_nnn = nnn × finalwgt_global). The dataset is saved in `pall`.

```
1184 * finalwgt

## [1] 1332.891

sum(pall$wgt_nnn) ; popsize  # match with the popsize = 1163650
```

```
## [1] 1163650

## [1] 1163650
```

For the missing seq, 0 weight is applied. Here's the summary.

```
head(pall, n = 3); tail(pall, n = 3)

##   geog1 sex agecode1 eth_code5 econg  nnn finalwgt_global      wgt_nnn piinv
## 1     1   1        1         1     1 1184        1.125753     1332.891  1184
## 2     1   1        1         1     2  190        1.125753      213.893   190
## 3     1   1        1         1     3 2103        1.125753     2367.458  2103
##   pp_seq
## 1      1
## 2      2
## 3      3

##      geog1 sex agecode1 eth_code5 econg nnn finalwgt_global     wgt_nnn  piinv
## 2518     6   2       14         5     1   0              NA     0.000000     0
## 2519     6   2       14         5     2   0              NA     0.000000     0
## 2520     6   2       14         5     3   2        1.125753     2.251505     2
##      pp_seq
## 2518   2518
## 2519   2519
## 2520   2520
```

Design matrix added

Now, using the pp table displaying a unique combination of patterns based on five variables, we build a design matrix. To achieve this, we will create dummy variables for each variable (column) and drop the last category, while keeping nnn (= piinv, raw count) and wgt_nnn. The intermediate results before renaming columns as des2-des26 are shown below. As demonstrated, the last category of the econg variable, econg_3 is dropped.

```
tail(design_matrix, n = 3)

##      geog1 sex agecode1 eth_code5 econg des1 geog1_1 geog1_2 geog1_3 geog1_4
## 2518     6   2       14         5     1    1       0       0       0       0
## 2519     6   2       14         5     2    1       0       0       0       0
## 2520     6   2       14         5     3    1       0       0       0       0
##      geog1_5 sex_1 agecode1_1 agecode1_2 agecode1_3 agecode1_4 agecode1_5
## 2518       0     0          0          0          0          0          0
## 2519       0     0          0          0          0          0          0
## 2520       0     0          0          0          0          0          0
##      agecode1_6 agecode1_7 agecode1_8 agecode1_9 agecode1_10 agecode1_11
## 2518          0          0          0          0           0           0
## 2519          0          0          0          0           0           0
## 2520          0          0          0          0           0           0
##      agecode1_12 agecode1_13 eth_code5_1 eth_code5_2 eth_code5_3 eth_code5_4
## 2518           0           0           0           0           0           0
## 2519           0           0           0           0           0           0
## 2520           0           0           0           0           0           0
##      econg_1 econg_2      wgt_nnn  piinv
```

```
## 2518       1       0     0.000000     0
## 2519       0       1     0.000000     0
## 2520       0       0     2.251505     2
```

To complete the function, these dummy variable columns (from gegog1_1 to econg_2) are renamed as `des1-des26`.

fn_des_pop_respmean()

The procedure is the same as seen earlier in 3.5. We now merge design matrix, popmean and respmean vectors.

fn_gh()

We repeat the procedure shown in 3.5.

fn_t_R_indicator()

The `fn_t_R_indicator()` allows us to compute propensity scores using matrix syntax. The computation of R-indicator relies on the propensity scores. As a preliminary step in the table-based R-indicator computation, the rsam and psam columns are being multiplied with the square root of `piinv`(represented by the column, of `gg`). Recall that `piinv` captures the cell total for each unique combination in the table.

```
head(gh[, c("seq", "gg", "piinv", " wgt_nnn")])
```

```
##   seq        gg piinv     wgt_nnn
## 1   1 34.409301  1184  1332.891152 = 1184*1.125753(finalwgt_global)
## 2   2 13.784049   190   213.893006
## 3   3 45.858478  2103  2367.457849
## 4   4  3.316625    11    12.383279
## 5   5  2.449490     6     6.754516
## 6   6  9.433981    89   100.191987
```

The corresponding syntax is provided below.

```
  px          <- as.matrix(gh[, psam_col])
  rx          <- as.matrix(gh[, rsam_col])
  gh   <- gh %>% mutate(gg = sqrt(piinv))
  gg   <- gh[, "gg"]
  pxm  <-  px * gg          # weighted_psam
  xxpm <-  t(pxm) %*% pxm
  rxm  <-  rx * gg          # weighted_rsam
  xxrm <-  t(rxm) %*% rxm
```

Subsequently, the function calculates propensity score, and the overall R-indicator.

We also compute weighted propensity scores by multiplying propensity scores, `roi`, by aggregate weights, `wgt_nnn`. Recall *wgt_nnn = piinv × finalwgt_global*.

```
gh_prop_mix <- cbind(gh, prop_mix) %>%
```

```
        mutate(roi = prop_mix) %>%
        mutate(roi_wgt = roi * wgt_nnn)
head(gh_prop_mix[, c("seq", var[1], "roi_wgt", "roi", "wgt_nnn") ])
##   seq geog1     roi_wgt       roi     wgt_nnn
## 1   1     1 1124.107366 0.8433602  1332.891152
## 2   2     1  178.256671 0.8333918   213.893006
## 3   3     1 2059.171606 0.8697817  2367.457849
## 4   4     1   10.459251 0.8446269    12.383279
## 5   5     1    5.637714 0.8346585     6.754516
## 6   6     1   87.272074 0.8710484   100.191987
```

R-indicator can be computed by 1 - standard deviation of propensity scores, or 1-sqrt(variance). In this example, the calculation would be 1-sqrt(0.06349736) as the variance of propensity scores is estimated as 0.06349736.

fn_t_rindicatorall()

The function computes the mean of weighted propensity scores across variables in loop. This is the sum of weighted propensity scores divided by the sum of final weights for each category of a variable. The equivalent expression using *one* variable without using loop is shown for illustration purposes. Please note the programme uses for loop function in R.

```
gh_prop_mix_ %>% group_by(geog1) %>%
  summarise(
   wgt_nnn_sum = sum(wgt_nnn),
   roi_wgt_sum = sum(roi_wgt) ) %>%
  mutate(
   roiwgtsum_dv_wgtnnnsum = roi_wgt_sum/wgt_nnn_sum )

## # A tibble: 6 × 4
## geog1   wgt_nnn_sum   roi_wgt_sum       roiwgtsum_dv_wgtnnnsum
##   <fct>       <dbl>       <dbl>                        <dbl>
## 1 1         155346.     186491.                        1.20
## 2 2         139651.     126262.                        0.904
## 3 3         147672.     150812.                        1.02
## 4 4         157456.     135449.                        0.860
## 5 5         160199.     214070.                        1.34
## 6 6         403327.     303761.                        0.753

186491.1 / 155346.0

## [1] 1.200489
```

The function also computes the overall mean of propensity scores.

```
mrphatall <- sum(df$roi_wgt) / sum(df$wgt_nnn)
mrphatall
## [1] 0.9597769
```

fn_partial()

The category and variable-level partical R-indicators can be otained by running `fn_partial()`.

The function concludes with exporting the results to a spreadsheet. For next steps, see section 3.5.

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.R` under the *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.

This concludes the manual. Thank you for taking the time reading the material. Please get in touch with any query or errata at demystify.stats@gmail.com.

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 (2025). “Quality Indicators for Administrative Data User Manual For R Package on GitHub, version 3.0”. Available at https://github.com/sook-tusk/qualadmin