Self-checking tools

Training and tools for checking data lab outputs

October 2022

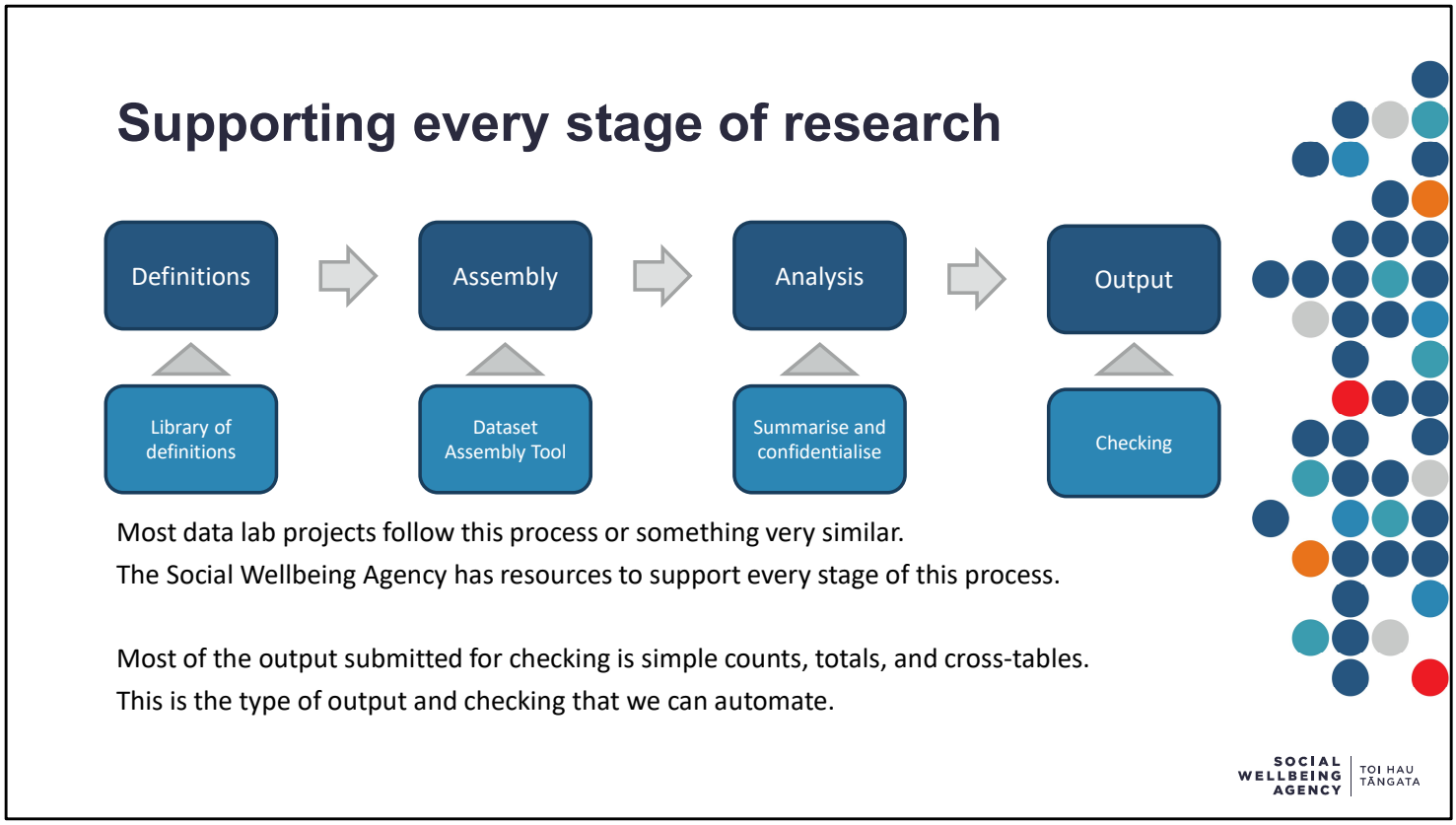
SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Author: Simon Anastasiadis

As part of lifting analytic and research capability, the Social Wellbeing Agency has developed a range of tools and resources to make research with integrated data more efficient. So that the wider government and research community can benefit, the Agency has made a practice of publishing these resources. Existing resources include the Dataset Assembly Tool, the IDI exemplar project, and tools for ease of summarising and confidentialising research outputs.

This documentation describes new tools to enable Integrated Data researchers to check their data lab outputs, including several worked examples. While Stats NZ are responsible for the process of releasing outputs from the secure environment, researchers are responsible for preparing outputs for submission and for the quality of those submissions. By having easy tools to check outputs prior to submission, researchers can ensure their outputs are of high quality. This will support a more efficient and timely checking service.

This presentation includes examples of code that were current at time of writing. ACTUAL CODE MAY HAVE BEEN UPDATED FOLLOWING PUBLICATION. Please see more recent resources if demonstrated code does not perform as expected.



Within the secure data lab environment, approved researchers can access unit record data for individuals and businesses. This data is stored in either the Integrated Data Infrastructure (IDI) or the Longitudinal Business Database (LBD). A key restriction for data lab research is that all outputs must be checked by Stats NZ before being released from the secure environment. This checking ensures that privacy and confidentiality is preserved.

To pass checking, all results need to be confidentialised. Most of the results created by research are simple counts, totals, and cross-tables. The Agency's summarise and confidentialise tools provide an effective way to produce this kind of output. The new checking tools described here provide an effective way to verify the confidentiality of this output.

These tools can also be used outside the data lab. But they have been designed with the data lab in mind and they reflect the confidentiality requirements of the IDI.

We recommend using this document alongside our other tools and documentation. Existing documentation can be found here: https://swa.govt.nz/publications/guidance/.

The code and accompanying files for these tools can be found on the Agency's GitHub page here: https://github.com/nz-social-wellbeing-agency. Versions of all those files are available inside the data lab, but for the latest version please see the Agency's website or GitHub page.

Where are we aiming for

1) Load code for tools into workspace

```
setwd("path/to/where/all/the/files/are/stored")
source("utility_functions.R")
source("summary_confidential.R")
source("check_confidentiality.R")
```

2) Load results from csv

```
conf_table = read.csv("path/file_name.csv", as.is = TRUE)
```

3) Ensure correct data types

```
library(dplyr)
conf_table = mutate(conf_table,
    raw_count = as.numeric(raw_count),
    raw_sum = as.numeric(raw_sum),
    conf_count = as.numeric(conf_count),
    conf_sum = as.numeric(conf_sum)
)
```

4) Specify individual checks

```
check_random_rounding(conf_table, "raw_count", "conf_count")
check_small_count_suppression(conf_table, "conf_count", 6, "raw_count")
check_small_count_suppression(conf_table, "conf_sum", 20, "raw_count")
check_absence_of_zero_counts(conf_table, "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

These sixteen lines of R code load a results file into R and check it to ensure the contents are ready for release. They make use of tools developed at the Social Wellbeing Agency to ensure output submissions are high quality.

This training documentation guides staff through the tools. It has been written to support the adoption and use of the Agency's tools. Working through this documentation will give researchers a strong understanding of this code and the confidence to adopt & adapt them in their own work.

All of the Agency's tools have been developed to work smoothly together. A more thorough example of the checking tools can be found in the IDI exemplar project, especially in the "`check_output - using R tools.R`" file.

The rest of this document works through each key component of the tools in detail.

Apply checking when ready to output

Output submission process

1. Create results for output using summarise tools
2. Confidentialise the results using confidentialisation tools
3. Write confidentialised results to file
4. **Optional – check confidentialisation of results**
5. Prepare file for output submission (including raw and confidentialised results)
    - Add coversheet & disclaimer
    - Add data dictionary
6. Duplicate prepared file and delete raw results
7. **Check confidentialisation of both files**
8. Submit files for output checking
    - Submit both files
    - Marking only the version without raw results as 'for release'

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

For most data lab outputs, researchers are encouraged to submit two files:

1. A file containing both raw and confidentialised results
2. A file containing only confidentialised results, with the raw results deleted.

The two files should be nearly identical. By providing two files, researchers support the checkers to validate how results were confidentialised.

In most cases, the easiest way to arrange this is to focus on preparing the first file. Once you are ready for submission, duplicate the file and delete the raw data so the second file is safe for release.

We recommend using these checking tools once both files are prepared for output submission.

Optionally, if you want to validate that our confidentialisation tools work correctly, you may wish to run checks on the files output by our confidentiality tools.

Stats NZ checkers have commented that SWA's template for output submissions is effective and easy to check. A copy of this template can be found under 'Resources' in the Exemplar project.

Loading tables for checking

Loading code for tools into workspace

```
setwd("path/to/where/all/the/files/are/stored")
source("utility_functions.R")
source("summary_confidential.R")
source("check_confidentiality.R")
```

Example reading csv file

```
conf_table = read.csv("path/file_name.csv", as.is = TRUE)
```

Example accessing sheet in Excel file

```
Library(readxl)
conf_table = read_xlsx("path/file_name.csv", sheet = "sheet_name")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The code for these tools is written in R. This means that to use this code some programming in R is required.

Throughout this presentation we give key lines of code to support researchers who are unfamiliar with R make use of these tools.

For researchers unfamiliar with R, the first step is to load your data into R. This slide demonstrates two approaches:

1. Reading from csv file
2. Reading from an Excel sheet

Before each of these approaches, we recommend you load the code for the tools into the workspace.

Loading a file as a csv tends to be more straightforward, as you are less likely to have additional formatting to interfere with the load (such as additional header lines, two tables on the same sheet, etc.).

For the most robust testing you will want to load a table that contains both the raw and confidentialised values. This enables the tools to compare raw and confidentialised values for consistency.

Establishing expected formats

Long-thin output format

- Columns-value pairs
- One row per summary
- Prefixes for raw and confidentialised columns

Example output:

col01	val01	col02	val02	summ.	raw_count	raw_sum	conf_count	conf_sum
Region	North			Income	25	35000	24	35000
Region	South			Income	15	45000	15	NA
Qual	Diploma	Region	North	Identity	11		12	
Qual	Degree	Region	North	Identity	14		15	
Qual	Degree	Region	South	Identity	4		NA	
Qual	Diploma	Region	South	Identity	11		9	

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Standard formats enable inputs to be manipulated by automated tools.

The most powerful of the tools we have created for checking require the table to be in long-thin format. This is the format produced and used by our summary and confidentialisation tools. Researchers who have results in a different format, will still be able to use some of our checking tools, but will not gain their full benefit.

The long-thin format we use is defined by:

- Pairs of "col" and "val" columns
    - The "col" columns contain the name of a column in the original dataset.
    - The "val" columns contain the unique values found in the corresponding column.
- A column for the variable summarised.
- Columns for the numeric result: number of distinct records, count of all records, and/or sum total of values.
- Numeric columns have raw_ or conf_ prefixes after they have been confidentialised.

Ensuring correct data types

Require numeric columns are stored as numbers

```
library(dplyr)
conf_table = mutate(conf_table,
    raw_count = as.numeric(raw_count),
    raw_sum = as.numeric(raw_sum),
    conf_count = as.numeric(conf_count),
    conf_sum = as.numeric(conf_sum)
)
```

Examine the loaded table

```
View(conf_table)
nrow(conf_table)
ncol(conf_table)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

When data is read into R, the software attempts to determine the data type of each column. As it does not always get this perfect, we may have to correct this before we can check the results.

The most likely correction we will need to make is ensuring columns that contain numeric results are stored as numbers. This most often occurs when the confidentialised table contains missing values, as these values might appear as the text "NA" or "S" or "NULL", and R sees some text values in the column and reads the entire column as text.

- `mutate()` is the command to change a column of a table, or create a new column.
- We use the `as.numeric()` command to convert text containing a number into a number.
- This command does not handle numbers with commas correctly. So best to save the numbers unformatted. For example as 1234 instead of as 1,234.
- Values that can not be converted to numbers will be replaced by NA (short for "Not Available"). R will give a warning message when this happens.

Some other useful options:

- We can also use `nrow()` and `ncol()` to check that R has read in the expected number of rows and columns.
- We can use `View(conf_table)` to look at the data in R (note the “V” is capitalised).

Specify every check

Check rounding to base 3

```
check_rounding_to_base_df(conf_table, conf_col)
```

Check graduated rounding

```
check_graduated_rounding_df(conf_table, conf_col)
```

Check rounding to base 3 and randomness of rounding

```
check_random_rounding(conf_table, raw_col, conf_col)
```

Check suppression where counts are small

```
check_small_count_suppression(conf_table, suppressed_col, threshold, count_col)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

In general, we recommend that researchers specify each check their data requires. This ensures the most thorough validation.

This slide shows the four main functions that can be used to run specific tests.

- Each of these tests returns TRUE if the test is passed and FALSE if the test is failed
- Some tests may also give warnings if the test is passed but is not perfect
- Some tests can also be set to print additional information to help the user identify any problems if they fail

We discuss each of these tests on the following slides.

Check rounding to base 3

Returns TRUE if the given column only contains values divisible by the base

```
check_rounding_to_base_df(
  df,           # the confidentialised table to check
  column,       # the name of column to check
  base = 3,     # values must be divisible by this number
  na.rm = TRUE # T/F whether missing values are ignored
)
```

Example use

```
check_rounding_to_base_df(conf_table, "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The first two inputs are compulsory. The last two are optional.

- Default values for the optional inputs are given after the equals sign.
- `base` allows the user to check for rounding to values other than 3.
- When `na.rm = TRUE`, missing values are ignored so only numeric values are checked.
- When `na.rm = FALSE`, missing values will also be checked. As missing values are not divisible by any number, the check will return FALSE if there are any missing values.

The result of the check can be:

- TRUE if all (non-missing) values in the column are divisible by base
- FALSE if a value in the column is found to not be divisible by base
- In addition, the check warns if all inputs are missing (NA)
- Alternatively, the check errors if the column is not found in the table or if the column is non-numeric.

Check graduated rounding

Returns TRUE if the column only contains value with graduated rounding

```
check_graduated_rounding_df(
  df,           # the confidentialised table to check
  column,       # the name of column to check
  na.rm = TRUE # T/F whether missing values are ignored
)
```

Example use

```
check_graduated_rounding_df(conf_table, "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The first two inputs are compulsory. The last one is optional.

- Default values for the optional input is given after the equals sign.
- When `na.rm = TRUE`, missing values are ignored so only numeric values are checked.
- When `na.rm = FALSE`, missing values will also be checked. As missing values are not divisible by any number, the check will return FALSE if there are any missing values.

The result of the check can be:

- TRUE if all (non-missing) values in the column are rounded as expected
- FALSE if a value in the column is found to not be rounded as expected
- In addition, the check warns if all inputs are missing (NA)
- Alternatively, the check errors if the column is not found or if the column is non-numeric.

Graduated random rounding rules in the Microdata Output Guide specify that values in a given range should be randomly rounded to a given base as follows:

values	base
0-18	3
19	2
20-99	5
100-999	10
1000+	100

Check random rounding

Returns TRUE if the given column only contains values divisible by the base and warns if rounding does not appear random

```
check_random_rounding(
  df,           # the confidentialised table to check
  raw_col,      # the name of column with raw values
  conf_col,     # the name of column with confidentialised values
  base = 3,     # values must be divisible by this number
  print_ratios = FALSE # T/F whether to display info on rounding
)
```

Example use

```
check_random_rounding(conf_table, "raw_count", "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The first three inputs are compulsory. The last two are optional.

- Default values for the optional inputs are given after the equals sign.
- `base` allows the user to check for rounding to values other than 3.
- This function automatically excludes missing values (equivalent to `na.rm = TRUE`).
- When `print_ratios = TRUE`, the check will also print out the proportion of values rounded up and down, and by how much. This can help explain why the check is warning about non-random rounding.

The result of the check can be:

- TRUE if all (non-missing) values in the column are divisible by base
- FALSE if a value in the column is found to not be divisible by base
- In addition, the check warns if:
    - all inputs are missing (NA)
    - raw/unrounded column is not provided
    - the difference between raw and rounded columns is too large
    - rounding does not appear to be random (e.g. all values rounded up)
- Alternatively, the check errors if the column is not found in the table or if the column is non-numeric.

Check suppression with small counts

Returns TRUE if the given column contains no values smaller than threshold

```
check_small_count_suppression(
  df,               # the confidentialised table to check
  suppressed_col,   # the column to check for suppression
  threshold,        # the smallest acceptable (raw) count
  count_col         # the name of the column with the raw count
)
```

Example use

```
check_small_count_suppression(conf_table, "conf_count", 6, "raw_count")
check_small_count_suppression(conf_table, "conf_sum", 20, "raw_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The first three inputs are compulsory. The last one is optional but recommended.

- If `count_col` is not provided, then we check `suppressed_col` against itself. This can be a useful check, but it is not a complete check, because suppression is based on raw counts and the suppressed column often also contains random rounding.

For some outputs you will have to run this check twice – once for counts and once for sums (note they use different thresholds, as per the example).
We expect: suppressed_col = NA if count_col < threshold.

The result of the check can be:

- TRUE if the suppressed column only has missing values when the count column is below the threshold.
- FALSE if any value in the suppressed column is non-missing when the count column is below the threshold
- Alternatively, the check errors if a required column is not found in the table or if the column is non-numeric.

Handling of zeros and small counts

Differences in the handling of zeros and small counts can pose a confidentiality risk

In the example table we can see:

- there are two types of qualification and three types of region
- two counts have been supressed (NA)
- Qual = Degree & Region = Central is not shown

col01	val01	col02	val02	summ.	conf_count
Qual	Diploma	Region	Central	Identity	NA
Qual	Diploma	Region	North	Identity	12
Qual	Diploma	Region	South	Identity	9
Qual	Degree	Region	North	Identity	15
Qual	Degree	Region	South	Identity	NA

From this we might deduce the suppressed values represent 1-5 people but there are zero people with Qual = Degree & Region = Central. This represents a confidentiality risk.

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

If small non-zero counts (often values 1-5) are suppressed and zero counts do not appear in the results, then there is a confidentiality risk that true/raw zeros could be recovered because they are handled differently from small non-zero counts.

In the example shown here:

- Small counts have been suppressed in the results.
- Zero counts have been omitted from the results.

This risk is more likely to be a concern when summarising small populations or very detailed results.

This risk should be checked for anytime a set of results contain suppressed values. It is easy to observe the presence of something (such as suppressed values) but harder to observe the absence of something (missing rows) without deliberate checks.

Check consistent handling of zeros

Returns TRUE if zero counts are clearly handled consistent with small counts

```
check_absence_of_zero_counts(
  df,                   # the confidentialised table to check
  conf_count_col,       # the column containing counts to be checked
  print_on_fail = FALSE # T/F whether to display missing rows on fail
)
```

Example use

```
check_absence_of_zero_counts(conf_table, "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

This function returns TRUE if it can be confident that the absence of zero counts does not pose a confidentiality risk. This means that either there are no suppressed counts in the table OR all combinations of labels/groups appear in the dataset. As a table or summarised results may be produced by appending multiple summaries the function checks within each combination. To use this function, the output needs to be in our expected format.

The first two inputs are compulsory. The last one is optional.

- Default value for the optional input is given after the equals sign.
- When `print_on_fail = TRUE`, then if the check fails it will print at least one combination that is absent from the dataset.

Note that the test fails if there are missing combinations in the results table. This can occur even if a researcher with subject matter knowledge knows that specific missing combinations do not pose a confidentiality risk. For example, a summary of region by urban/rural description will fail this check as not every region contains a major urban centre. A researcher with subject matter knowledge will know that not all combinations of region and urban/rural description should appear in their output. But the checking tool does not have this subject matter knowledge. In these cases you may have to examine the file manually.

If zeros & small counts are not consistent

Option 1 – remove suppressed small counts from the results

```
conf_table = filter(conf_table, !is.na(conf_count))
```

Option 2 – expand results to include zeros for all missing rows

```
conf_table = expand_to_include_zero_counts(conf_table)
# requires conf_table to be in our expected format
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

If a risk is identified `check_absence_of_zero_counts` returns FALSE, then there are two common solutions:

1. remove all rows that contain suppression of small non-zero counts
2. use `expand_to_include_zero_counts` to add zero count rows into dataset

Option 1 treats small counts the same as non-zero counts.

- `filter()` keeps only those rows that pass the condition(s)
- `is.na()` returns TRUE if a value is missing, and `!` is the logical not, so `!is.na()` returns true if a value is not missing.

Option 2 treats zero counts the same as small counts.

- `expand_to_include_zero_counts` works within each summary combination to creates additional rows where a combination of labels/groups was missing.
- To use this function, the output needs to be in our expected format.

We recommend

- option 1 when the results are large or there are many missing values (the output file will be smaller),
- option 2 when there are only a small number of zero values omitted from the results.

Putting it all together

1) Load code for tools into workspace

```
setwd("path/to/where/all/the/files/are/stored")
source("utility_functions.R")
source("summary_confidential.R")
source("check_confidentiality.R")
```

2) Load results from csv

```
conf_table = read.csv("path/file_name.csv", as.is = TRUE)
```

3) Ensure correct data types

```
library(dplyr)
conf_table = mutate(conf_table,
    raw_count = as.numeric(raw_count),
    raw_sum = as.numeric(raw_sum),
    conf_count = as.numeric(conf_count),
    conf_sum = as.numeric(conf_sum)
)
```

4) Specify individual checks

```
check_random_rounding(conf_table, "raw_count", "conf_count")
check_small_count_suppression(conf_table, "conf_count", 6, "raw_count")
check_small_count_suppression(conf_table, "conf_sum", 20, "raw_count")
check_absence_of_zero_counts(conf_table, "conf_count")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

This code provides an end-to-end demonstration of the checking tools.

1. Load code for tools into workspace
2. Load results from csv file
3. Ensure data has correct types
4. Specify and run individual checks

If you are using this as a template for your own analysis, then the key places to edit are:

- The path to where files are stored
- The name of your result file

A more complete worked example can be found in the Exemplar project, in the file check_output - using R tools.R (which is located in the Output folder).

Note that you may still need to run additional checks, such as entity counts, once these checks are complete. We have only automated the most common checks.

Bonus: Overview of result contents

For getting an overview of results prior to submitting for output

```
explore_output_report(
  df,                 # the confidentialised table to check
  output_dir,         # the directory to save to
  output_label        # an additional label for the report(s)
)
```

Example use

```
explore_output_report(conf_table)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

It can be very frustrating to output results from the secure data lab environment only to discover shortly after release that they contain obvious errors. One way to reduce the risk of this is to review the results prior to release.

The `explore_output_report` function produces up to three reports – one for each of the possible output columns (distinct, count, and sum). These reports give an overview of each measure to help the researcher confirm the values are reasonable. To use this function, the output needs to be in our expected format.

The first input is compulsory. The last two are optional.

- `output_dir` enables the research to specify a folder to write the reports to.
- `output_label` enables the researcher to specify an extra label for the reports, this is recommended if reporting on multiple tables

It is expected that researchers will know what reasonable values are for each variable and hence will be able to confirm from reviewing such a summary that the outputs are as expected.

Note that values may fluctuate depending on the subgroups used. For example, if there are results for both all New Zealand and for **only** those people who are recent migrants, then the range of reasonable values should be wider than if results included only the entire population or only recent migrants.

Bonus: Check with a single command

Run all the common checks with a single command "check_confidentialised_results"

```
msg = check_confidentialised_results(conf_table)
print(msg)                  # show report
cat(paste(msg,"\n"))        # show report, looks nicer
```

Get a report of tests passed, failed, or skipped

```
conf_distinct        checked for RR3 : SKIP
conf_distinct  suppressed if raw < 6 : SKIP
   conf_count        checked for RR3 : PASS
   conf_count  suppressed if raw < 6 : PASS
     conf_sum suppressed if raw < 20 : PASS
          all absence of zero counts : PASS
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

For best results, researchers should give thought to what checks are required to ensure their output meets confidentiality standards and to run those checks. However, in some cases, it may be convenient to run the most common checks in a single step. We have written the `check_confidentialised_results()` function to run a set of common checks. To use this function, the output needs to be in our expected format.

The results message with report on how the table performed against each test:

- `SKIP` = the test could not be run
- `PASS` = test runs and output meets expectations
- `FAIL` = test runs and output does not meet expectations

The most common reason for a test to skip is that the columns required to run the test are not available in the confidentialised table. For example, if `conf_table` does not contain the columns `raw_distinct` and `conf_distinct` then we can not test to see if `conf_distinct` has been randomly rounded. Hence this test is skipped.

Thank you for your time and attention.

The checking tools demonstrated above have improved the quality of output submissions by Social Wellbeing Agency staff.

We encourage you to adopt these methods so that you might also create higher quality output submissions. This will make checking your output submission easier, supporting a more efficient and timely checking service.

For further guidance on preparing effective output submissions, please also see the examples in the IDI exemplar project.