Summarise and confidentialise

Tools for producing outputs from the data lab

October 2022

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

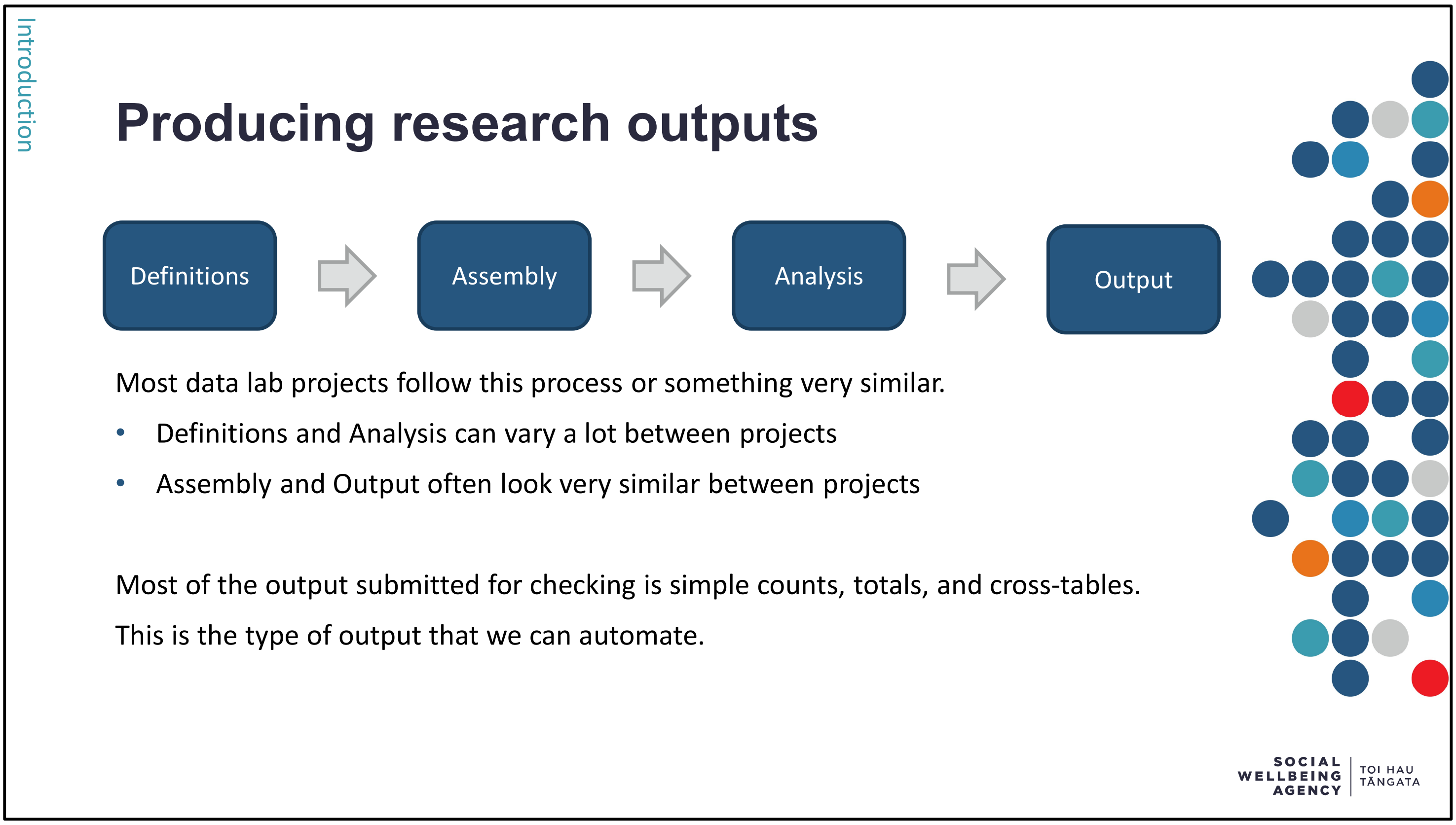
Author: Simon Anastasiadis

In December 2020, the Social Wellbeing Agency published the Dataset Assembly Tool: A resource to make data wrangling faster and easier. Following the development of this tool, the Agency developed new tools to summarise and confidentialise research outputs. The code that provides the new tools is available alongside the Dataset Assembly Tool. This presentation provides guidance on how the summarise and confidentialise tools work, including several worked examples.

Both the Dataset Assembly Tool and these new tools are demonstrated in the IDI exemplar project – published by the Agency in April 2022. We recommend using this document alongside the Dataset Assembly Tool and the IDI exemplar project documentation. Both sets of documentation can be found here: https://swa.govt.nz/publications/guidance/.

The code and accompanying files 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.

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, provided by Stats NZ, 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 the privacy and confidentiality is preserved.

This checking requirement also effects how research using the IDI and LBD is delivered: It creates a distinct Output stage where results need to be summarised and confidentialised. This is the stage of a research project that these tools are designed for.

Most of the output created in this stage and submitted to Stats NZ for checking is simple counts, totals, and cross-tables. This is the output type that these tools are designed for.

These tools can be used outside the data lab environment – they work with any dataset in R, and can also work with data stored in a database. But they have been designed with the data lab in mind and they reflect the confidentiality requirements of the IDI.

Introduction

Where we are aiming for

1) Load code for tools into workspace

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

2) Load a csv file

```
rectangular_table = read.csv("./path/input_file_name.csv", as.is = TRUE)
```

3) Columns to group by

```
my_cols = c("Qual", "Region", "Age")
all_groups = cross_product_column_names(my_cols, my_cols)
```

4) Summarise

```
output = summarise_and_label_over_lists(
  df = rectangular_table,
  group_by_list = all_groups,
  summarise_list = list("Identity"),
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE)
```

5) Confidentialise

```
conf_output = confidentialise_results(output)
```

6) Write summary to csv

```
write.csv(conf_output, "./path/output_file_name.csv")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

These fifteen lines of R code make summarised and confidentialised results ready for output submission. They make use of tools developed at the Social Wellbeing Agency to save hours of effort by researchers.

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 these fifteen lines 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 full, end-to-end example of the use of these tools can also be found in the IDI exemplar project, especially in the "`output_results - using R tools.R`" file.

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

Setup

Establishing expected formats

Rectangular input format

- One row per identity/person/business
- One column per measure

Long-thin output format

- Columns-value pairs
- One row per summary

Example input:

Identity	Qual	Region	Income
1001		North	$5000
1002	Diploma	North	$15000
1003	Degree	North	$15000
1004	Degree	South	
1005	Diploma	South	$20000
1006	Diploma	South	$25000

Example output:

col01	val01	col02	val02	summ.	count	sum
Region	North			Income	3	35000
Region	South			Income	2	45000
Qual	Diploma	Region	North	Identity	1	
Qual	Degree	Region	North	Identity	1	
Qual	Degree	Region	South	Identity	1	
Qual	Diploma	Region	South	Identity	2	

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Expected or required formats enable inputs to be manipulated by automated tools.

There is no required format for the input table. The tool does not impose any restriction on the column names, number of rows, or the number of columns. But it has been designed anticipating that most input tables will have one row per identity (person, business, household – in the IDI this is often snz_uid) and one column per measure. In the left example, the table gives the identity number (ID), qualification, region, and income for six people.

The summarising tool produces output according to a pre-defined long-thin format. The confidentialisation tool requires its input to take this format and outputs this format. The format is defined by:

- Pairs of "col" and "val" columns
    - The "col" columns contain the name of a column in the rectangular 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.

Loading data for summarising

Loading code for tools into workspace

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

Example accessing database table

```
db_con = create_database_connection(database = "IDI_Sandpit")
rectangular_table = create_access_point(db_con,"[db]", "[schema]", "[table]")
```

Example reading csv file

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

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. Accessing a table stored in the database
2. Reading an existing csv file

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

When working with large input tables, accessing the table from a database is recommended. There are limits on the memory available in R, but when accessing a table this way it can be read from disk as required.

Loading a file as a csv tends to be more straightforward for researchers who prefer other programming languages as most languages include options to output data tables to csv files.

Single summary

Run a single summary

The core function to produce a single summary is "summarise_and_label"

```
summarise_and_label(
    df,                  # the rectangular dataset to summarise
    group_by_cols,       # the names of the columns to group by
    summarise_col,       # the name of the column to summarise
    make_distinct,       # T/F whether to count distinct values
    make_count,          # T/F whether to count values
    make_sum,            # T/F whether to sum values
    clean,               # options for cleaning the column to summarise
    remove.na.from.groups # T/F should missing values be removed
)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Researchers familiar with SQL or SAS will recognise this typical approach to summaries:

```
SELECT col1, col2, COUNT(*) AS num, SUM(col4) AS sum
FROM rectangular_table
GROUP BY col1, col2
```

Our core function works in much the same way, but with some features to make multiple summaries easier. Use of this function is demonstrated on the next two slides.

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

- T/F inputs need to take the value TRUE or the value FALSE (in R these are typed in capitals)
- `remove.na.from.groups` defaults to TRUE, this means that missing values in the group by columns are removed
- `clean` can taken any of the values `"none"`, `"na.as.zeros"`, or `"zeros.as.na"`.
    - The default value is `"none"` and makes no changes
    - `"na.as.zeros"` replaces missing values with zeros
    - `"zeros.as.na"` replaces zeros with missing values

The summary tool can produce three types of outputs: counts of the number of distinct values, counts of the number of non-missing values, and the sum of the values. We do not have an option to produce a mean or average. The confidentiality rules require that means are produced with the confidentialised counts. This is a more complex rule to apply or check. So instead, the tool produces counts and sums separately. These can be checked for confidentiality separately and combined to produce a mean afterwards.

Single summary

Single summary – example 1

```
output = summarise_and_label(
  df = rectangular_table,
  group_by_cols = c("Qual", "Region"),
  summarise_col = "Identity",
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE)
```

Identity	Qual	Region	Income
1001		North	$5000
1002	Diploma	North	$15000
1003	Degree	North	$15000
1004	Degree	South	
1005	Diploma	South	$20000
1006	Diploma	South	$25000

col01	val01	col02	val02	summ.	count
Qual	Diploma	Region	North	Identity	1
Qual	Degree	Region	North	Identity	1
Qual	Degree	Region	South	Identity	1
Qual	Diploma	Region	South	Identity	2

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table on the left, the code above creates the table shown on the right and stores it in a variable called `output`. Note how the columns specified in `group_by_cols` and `summarise_col` turn up as values in the output table.

This R code works very similar to the following SQL code:

```
SELECT Qual, Region, COUNT(Identity) AS count
FROM rectangular_table
GROUP BY Qual, Region
```

For every combination of qualification and region, we count the number of people (identities).

When inputting multiple groups they need to be included within `c()` and separated by commas. For example: `c("col1", "col2", "col3")`. The quote marks tell R that these inputs are text (character strings), which is the type of input the tool is expecting.

Because we have not specified the value of `remove.na.from.groups` it has defaulted to TRUE. This setting means that the Qual = missing, Region = North combination is not part of the output. To includes this combination in the output we would need to specify `remove.na.from.groups = FALSE`.

Single summary

Single summary – example 2

```
output = summarise_and_label(
  df = rectangular_table,
  group_by_cols = "Region",
  summarise_col = "Income",
  make_distinct = FALSE, make_count = TRUE, make_sum = TRUE)
```

Identity	Qual	Region	Income
1001		North	$5000
1002	Diploma	North	$15000
1003	Degree	North	$15000
1004	Degree	South	
1005	Diploma	South	$20000
1006	Diploma	South	$25000

col01	val01	summ.	count	sum
Region	North	Income	3	35000
Region	South	Income	2	45000

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table on the left, the code above creates the table shown on the right and stores it in a variable called `output`. Note how we set both `make_count` and `make_sum` to be TRUE and this produces two output columns.

This R code works very similar to the following SQL code:

```
SELECT Region, COUNT(Income) AS count, SUM(Income) AS sum
FROM rectangular_table
GROUP BY Region
```

For every region, we count the number of people (identities) and sum their income.

As mentioned on slide 6, the summary tool does not output means or averages. When we want a mean, we instead create the count and the sum (like this example) and calculate the mean after applying confidentialisation.

Because there is only one `group_by_col` in the code, the output includes only one pair of col & val columns. If we were to join this output table with the previous output table (using the command `bind_rows`), then empty col02 and val02 columns would be added to this table so the columns aligned.

Missing values are not counted. This is why the count for region = South takes the value 2. If we had set `clean = "na.as.zeros"` then this missing value would have been replaced by a 0 during the calculation and the count for Region = South would have been 3 instead.

Single summary

Use cleaning options when summarising

`clean` and `remove.na.from.groups` change how missing values are handled

When calculating the number of distinct values or a count of all values, the standard behaviour in SQL is that missing values are not counted.

Missing values in SQL are denoted as `NULL`. Missing values in R are denoted as `NA` (“Not Available”)

`clean` controls the handling of missing values in the summarised columns
- Zero values can be converted to missing, or missing can be converted to zero.

`remove.na.from.groups` controls the handling of missing values in the group by columns
- Groups without a label can be automatically removed from the output.

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The `clean` input controls handling of zeros and missing values in the summarised columns.
- It is common for data to contain missing values, and it can be cumbersome to set all missing values to zero. We can use `clean = "na.as.zeros"` to replace all missing values with zeros during the summarising. This ensure that missing values are included when counting records.
- When working with binary indicator variables, zero values often indicate the absence of an attribute. When summarising, we may not want to count zero values. Hence using `clean = “zeros.as.na"` will replace zeros with missing values during the summarising. This ensures the zero values are excluded when counting records.
- Another common application for this control arises when considering average income. If you want to calculate the average income of only those people with income, then using `clean = “zeros.as.na"` will ensure people without income will be excluded. If you want to calculate the average income of all people (whether or not they had income), then using `clean = "na.as.zeros"` will ensure all people are included.

The `remove.na.from.groups` input controls how missing values in the grouping columns are handled. For example, when producing a regional summary how should we summarise people who do not have a region.
- With `remove.na.from.groups = FALSE` the summarised output **will** contain a row for people who do not have a region.
- With `remove.na.from.groups = TRUE` the summarised output **will not** contain a row for people who do not have a region.

Run multiple summaries

Multiple summaries

The function to produce a multiple summaries is "summarise_and_label_over_lists"

```
summarise_and_label_over_lists(
    df,                 # the rectangular dataset to summarise
    group_by_list,      # lists of the columns to group by
    summarise_list,     # lists of the column to summarise
    make_distinct,      # T/F whether to count distinct values
    make_count,         # T/F whether to count values
    make_sum,           # T/F whether to sum values
    clean,              # options for cleaning the column to summarise
    remove.na.from.groups # T/F should missing values be removed
)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Producing a single summary is straightforward in most programming languages and does not require special tools. The key advantage of our summary tool is how it produces multiple summaries with a single command.

The core function to produce a multiple summaries is called `summarise_and_label_over_lists`. Use of this function is demonstrated on the next two slides.

This function takes almost identical inputs to the single summary function. The key difference is that multiple sets of grouping columns and multiple summarise columns can now be specified.

Multiple summaries – example 1

```
output = summarise_and_label_over_lists(
  df = rectangular_table,
  group_by_list = list("Qual", "Region"),
  summarise_list = list("Identity", "Income"),
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE)
```

Identity	Qual	Region	Income
1001		North	$5000
1002	Diploma	North	$15000
1003	Degree	North	$15000
1004	Degree	South	
1005	Diploma	South	$20000
1006	Diploma	South	$25000

Col01	val01	summ.	count
Qual	Diploma	Identity	3
Qual	Degree	Identity	2
Region	North	Identity	3
Region	South	Identity	3
Qual	Diploma	Income	3
Qual	Degree	Income	1
Region	North	Income	3
Region	South	Income	2

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table on the left, the code above creates the table shown on the right and stores it in a variable called `output`. Note how the output includes all the combinations of the columns given in the inputs.

Overall, four summaries are produced and then stacked. For each type of qualification and region, we count both the number of people (identities) and the number with income. For example, there are two people with a degree qualification but only one person with a degree qualification has income.

When `summarise_and_label_over_lists` is used, it runs `summarise_and_label` multiple times and stacks all the output into a single table.

You can then give this output directly to the confidentialisation function (demonstrated in a later slide) or write it out to csv.

We often use the Identity column when counting as we want to count every single row, and this column is guaranteed to have a value in every single row. We do not recommend making the sum of the Identity column – when working with a database, the sum of ID numbers can result in numbers that are too large for the data type, leading to errors.

Multiple summaries – example 2

```
output = summarise_and_label_over_lists(
  df = rectangular_table,
  group_by_list = list(c("Qual", "Region"), "Region"),
  summarise_list = list("Identity"),
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE)
```

Identity	Qual	Region	Income
1001		North	$5000
1002	Diploma	North	$15000
1003	Degree	North	$15000
1004	Degree	South	
1005	Diploma	South	$20000
1006	Diploma	South	$25000

col01	val01	col02	val02	summ.	count
Qual	Diploma	Region	North	Identity	1
Qual	Degree	Region	North	Identity	1
Qual	Diploma	Region	South	Identity	2
Qual	Degree	Region	South	Identity	1
Region	North			Identity	3
Region	South			Identity	3

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table on the left, the code above creates the table shown on the right and stores it in a variable called `output`. Note how the output includes all the combinations of the columns given in the inputs.

Overall, two summaries are produced and then stacked. For the first summary, we count the number of people with each combination of qualification and region (e.g. 'degree' and 'north' has 1 person). For the second, we just count the number of people in each region (e.g. 'north' has 3 people).

Note that `group_by_list` and `summarise_list` need to be given lists.

- This is why even though there is only a single value to summarise, it is placed inside `list()`
- As this example shows, we can combine multiple columns inside a list using `c()`

Two separate summaries must be done to complete this command. Because the Region summary will have fewer output columns than the Qual and Region summary, when the two summaries are combined into a single output we have empty cells in the `col02` and `val02` columns.

Quick setup for multiple summaries

Create groups

We created "cross_product_column_names" to quickly produce groups

```
cross_product_column_names(
    ...,                   # sets of column names inside c()
    always,                # columns to include in all groups
    drop.dupes.within,     # T/F remove duplicates within output
    drop.dupes.across      # T/F remove duplicates across output
)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Often we want to produce many different summaries all at once. Rather than needing to type each of these combinations (which would be very time consuming), we created `cross_product_column_names` to automatically produce combinations of its inputs. Use of this function is demonstrated on the next few slides.

The `...` input can take any number of sets of column names. Each set is contained inside `c()` and separated by commas between sets. The function will output groups that are one column from every set, covering all possible combinations.

The last three inputs are optional.

- `always` allows you to specify columns that should be used in every summary. By default it includes no columns in every output.
- `drop.dupes.within` defaults to TRUE. When TRUE the function checks for and removes cases where the same column appears more than once in a single group. For example: (Qual, Region, Region) would become (Qual, Region). With this setting turned off the summary tool is likely to error.
- `drop.dupes.across` defaults to TRUE. When TRUE the function checks for and removes equivalent groupings across all groupings. For example: it would ensure you received only one of (Qual, Region) and (Region, Qual). If you want to allow for different orders of the same columns set it to FALSE.

Create groups

Controls determine handling of duplicates

Example output with different settings of `drop.dupes` controls

```
drop.dupes.within = FALSE, drop.dupes.across = FALSE
      c("Qual", "Qual"), c("Qual", "Region"), c(Region", "Qual")

drop.dupes.within = TRUE, drop.dupes.across = FALSE
      c("Qual"),         c("Qual", "Region"), c(Region", "Qual")

drop.dupes.within = FALSE, drop.dupes.across = TRUE
      c("Qual", "Qual"), c("Qual", "Region")

drop.dupes.within = TRUE, drop.dupes.across = TRUE
      c("Qual"),         c("Qual", "Region")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Duplicates can appear within a single grouping or across groupings. This slide gives four examples of output with each combination of `drop.dupes.within` and `drop.dupes.across`.

In each example, a set of column names is contained within a `c()`. This is how groups of things are coded in R. So `c("Qual", "Region")` denotes a group in R containing both Qual and Region (as text strings).

1. The first case, where both values are FALSE, can be considered the raw output. Note that Qual appears twice within the first group, and that the second and third groups are equivalent, but in different orders.
2. The second case has `drop.dupes.within = TRUE`. Note that Qual no longer appears twice within the first group. But the second and third groups are still equivalent.
3. The third case has `drop.dupes.across = TRUE`. Note that the third group has been removed as it was equivalent to the second group. But the first group still contains Qual twice.
4. The fourth case has both values set to TRUE. Note that Qual only appears once in the first group, and the third group has been removed.

For most applications, researchers will want `drop.dupes.within = TRUE`. But you will have to choose `drop.dupes.across` depending on whether you want different orders of the group by columns .

Create groups

Quick setup for groups – example 1

Manual

```
all_groups = list(
    c("Qual"),          c("Qual", "Region"), c("Qual", "Age"),
    c(Region", "Qual"), c("Region"),         c("Region", "Age"),
    c("Age", "Qual"),   c("Age", "Region"),  c("Age")
)
```

Using "cross_product_column_names"

```
all_groups = cross_product_column_names(
    c("Qual", "Region", "Age"), c("Qual", "Region", "Age"),
    drop.dupes.across = FALSE, drop.dupes.within = TRUE)
```

Using "cross_product_column_names" v2

```
my_cols = c("Qual", "Region", "Age")
all_groups = cross_product_column_names(my_cols, my_cols,
    drop.dupes.across = FALSE , drop.dupes.within = TRUE)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The three examples above produce identical output: a variable called `all_groups` that is a list with nine different combinations of Qual, Region, and Age. This variable can be input directly into `summarise_and_label_over_lists` as the `group_by_list`.

While we can do this manually (like the first case) this quickly becomes cumbersome as the number of groups increases. The second case is faster and easier. It takes two sets of column names and produces all pairwise combinations between them.

In the third case, instead of entering the columns directly, we store the list of column names in a variable called `my_cols` and give this as an input. Because we give this twice, the output includes pairwise combinations.

Because we set `drop.dupes.across = FALSE` the output includes both orders of the variables. In this example, we have both `c("Qual", "Age")` and `c("Age", "Qual")`. If we had used the default `drop.dupes.across = TRUE` then only one of each pair would have been included in the output.

When doing all pairwise combinations of the input columns we might expect the output to include `c("Qual", "Qual")` this output is replaced by `c("Qual")` because of the setting `drop.dupes.within = TRUE`.

Create groups

Quick setup for groups – example 2

Manual

```
all_groups = list(
    c("Qual", "Age", "Ethnicity"),
    c("Region" , "Age", "Ethnicity")
)
```

Using "cross_product_column_names"

```
all_groups = cross_product_column_names(
    c("Qual", "Region"),
    always = c("Age", "Ethnicity"))
```

Using "cross_product_column_names" v2

```
my_cols = c("Qual", "Region")
always_cols = c("Age", "Ethnicity")
all_groups = cross_product_column_names(my_cols, always = always_cols)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The three examples above produce equivalent output: a variable called `all_groups` that is a list with different combinations of Qual, Region, Age, and Ethnicity. This variable can be input directly into `summarise_and_label_over_lists` as the `group_by_list`.

In the third case, instead of entering the columns directly, we store the group column names in a variable called `my_cols` and the always columns in a variable called `always_cols`. These two variables are given as inputs to `cross_product_column_names`.

Note that we have to specify "`always =`" for the `always` input. These two inputs will produce different outputs:

- `cross_product_column_names(my_cols, always = always_cols)`
- `cross_product_column_names(my_cols, always_cols)`

In the second case (without "`always =`"), the function will instead produce all pairs where the first column comes from `my_cols` and the second column comes from `always_cols`.

Combining groupings

Create groups

Naïve approach – also produces unwanted pairs of ethnic groups

```
demographic_columns = c(
    "Age", "Sex", "Eth_Euro", "Eth_Maori", "Eth_Pacific", "Eth_Asian"
)
all_groups = cross_product_column_names(demographic_columns, demographic_columns)
```

Refined approach – does not produce unwanted pairs of ethnic groups

```
demographic_columns = c("Age", "Sex")
ethnicity_columns = c("Eth_Euro", "Eth_Maori", "Eth_Pacific", "Eth_Asian")
all_groups = c(
    cross_product_column_names(demographic_columns, demographic_columns),
    cross_product_column_names(demographic_columns, ethnicity_columns)
)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Suppose we want to produce pairwise combinations of all demographics columns (age, sex, and ethnicity). However, as we are using full response ethnicity, people can have more than one ethnicity, and ethnicity is stored in several different columns.

Two examples for handling ethnicity are shown above.

The first example is naive. Because we include all ethnicity columns together with the non-ethnicity columns (age and sex), the resulting pairs of columns will also contain pairs of ethnicities. For example `c("Eth_Euro", "Eth_Maori")`. For most analyses we will not want pairwise combinations of ethnicity columns, so this approach creates extra output that is unwanted.

A better approach would be to separate the ethnicity columns from the other demographic columns. We can then produce two cross-products. The first will produce all pairs of the non-ethnicity demongraphic columns, and the second will produce all pairs of ethnicity with one of the non-ethnicity columns. Just as we can use `c()` to combine together the names of columns, we can use it to combine together the pairs of column names produced by `cross_product_column_names`.

Confidentialise

Confidentialising results

The core function to confidentialising is "confidentialise_results"

```
confidentialise_results(
    df,                 # summarised output to confidentialise
    stable_RR,          # T/F enforce consistent rounding
    sum_RR,             # T/F apply random rounding to the sum
    BASE,               # the base for random rounding
    COUNT_THRESHOLD     # the minimum value for counts
    SUM_THRESHOLD       # the minimum count for sums
)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The core function to confidentialise summaries is called `confidentialise_results`. Use of this function is demonstrated on the two next slides.

With the default settings, this function applies standard confidentiality checks for IDI outputs:

- Counts are randomly rounded to base 3
- Counts less than 6 are suppressed
- Sums/totals from less than 20 individuals are suppressed

Only the first input is compulsory. All the others are optional, and default to standard IDI values.

- `stable_RR` determines whether random rounding should be consistent between runs. By default it is FALSE, because it is much slower to have this set to TRUE.
- `sum_RR` determines whether the sum/total should also be randomly rounded. By default it is FALSE, but for some outputs you may set it to TRUE. (For example, when confidentialising total income, there is no need to random round. But if you had summed number of children per person to produce total number of children, then you might want to apply random rounding to this total.)
- `BASE` determines the number we randomly round to. By default it is 3.
- `COUNT_THRESHOLD` determines the minimum raw count that is not suppressed. By default it is 6.
- `SUM_THRESHOLD` determines the minimum raw count so that totals are not suppressed. By default it is 20.

Confidentialising – example 1

Confidentialise

col01	val01	col02	val02	summ.	count
Qual	Diploma	Region	North	Identity	10
Qual	Degree	Region	North	Identity	10
Qual	Degree	Region	South	Identity	5
Qual	Diploma	Region	South	Identity	29

```
conf_output = confidentialise_results(output)
```

col01	val01	col02	val02	summ.	raw_count	conf_count
Qual	Diploma	Region	North	Identity	10	9
Qual	Degree	Region	North	Identity	10	12
Qual	Degree	Region	South	Identity	5	NA
Qual	Diploma	Region	South	Identity	29	28

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table at the top, the code in the middle creates the table shown at the bottom and stores it in a variable called `conf_output`.

Notes:

- The original numeric column(s) are given the prefix "raw_"
- The confidentialised numeric column(s) are given the prefix "conf_"
- Suppressed values appear as NA. In R this means 'Not Available' and is how missing values are shown.

`confidentialise_results` applies both random rounding and small count suppression. Our tools include functions for applying only random rounding or small count suppression. These are discussed below.

Confidentialising – example 2

Confidentialise

col01	val01	summ.	distinct	count	sum
Qual	Diploma	Income	19	20	12345
Qual	Degree	Income	20	23	23456
Region	North	Income	5	8	345
Region	South	Income	26	29	98765

```
conf_output = confidentialise_results(output)
```

col01	val01	summ.	raw_distinct	raw_count	raw_sum	conf_distinct	conf_count	conf_sum
Qual	Diploma	Income	19	20	12345	18	21	NA
Qual	Degree	Income	20	23	23456	21	24	23456
Region	North	Income	5	8	345	NA	9	NA
Region	South	Income	26	29	98765	27	27	98765

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Given the rectangular table at the top, the code in the middle creates the table shown at the bottom and stores it in a variable called `conf_output`.

Notes:
- The distinct and count columns are confidentialised separately.
- At least one of the distinct and count columns must be present to confidentialise the sum column.
- If both the distinct and count columns are present, then the sum will be suppressed if either is less than 20.
- Raw counts of 20 are forced to round up to protect confidentiality (if rounded down to 18 then the original value can be deduced because we know that totals are only released if the count is at least 20).

Component functions for confidentialising

For when `confidentialise_results` is not suitable

```
# Randomly rounds a numeric vector to the given base.
randomly_round_vector(input_vector, base = 3, seeds = NULL)

# Applied random rounding to specified columns of a dataset.
apply_random_rounding(df, RR_columns, BASE = 3, stable_across_cols = NULL)

# Applied graduated random rounding (GRR) to specified columns of a dataset.
apply_graduated_random_rounding(df, GRR_columns, stable_across_cols = NULL)

# Suppresses values where the count is too small.
apply_small_count_suppression(df, suppress_cols, threshold, count_cols = suppress_cols)
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

The `confidentialise_results` function applies all the standard confidentialisation rules. Researchers wanting finer control over the confidentialisation process may want to use the component subfunctions that are part of our tools. A short list of these subfunctions is given above. We recommend that researchers use the `confidentialise_results` function where possible, and only use the component subfunctions if additional confidentialisation is required.

Key points to note if using the component functions:

- Inputs with an equals sign after them are optional and have the default value given by the equals sign.
- `apply_random_rounding` works on dataframes, `randomly_round_vector` does not work on dataframes (it only works on vectors).
- `apply_graduated_random_rounding` applies the GRR proceedure to a dataframes specified in the Microdata Output Guide by Stats NZ. For other forms of graduated rounding you will need to combinine `randomly_round_vector` and `case_when`.
- `apply_small_count_suppression` applies suppression to a dataframe. It can take both a column to suppress and a column that contains the number of observation (count).

More detailed information on each function can be found in the summary_confidential.R file. Before each function in this file is a text description of the function and what it does.

Conclude

Combining it all together

1) Load code for tools into workspace

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

2) Load a csv file

```
rectangular_table = read.csv("./path/input_file_name.csv", as.is = TRUE)
```

3) Columns to group by

```
my_cols = c("Qual", "Region", "Age")
all_groups = cross_product_column_names(my_cols, my_cols)
```

4) Summarise

```
output = summarise_and_label_over_lists(
  df = rectangular_table,
  group_by_list = all_groups,
  summarise_list = list("Identity"),
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE)
```

5) Confidentialise

```
conf_output = confidentialise_results(output)
```

6) Write summary to csv

```
write.csv(conf_output, "./path/output_file_name.csv")
```

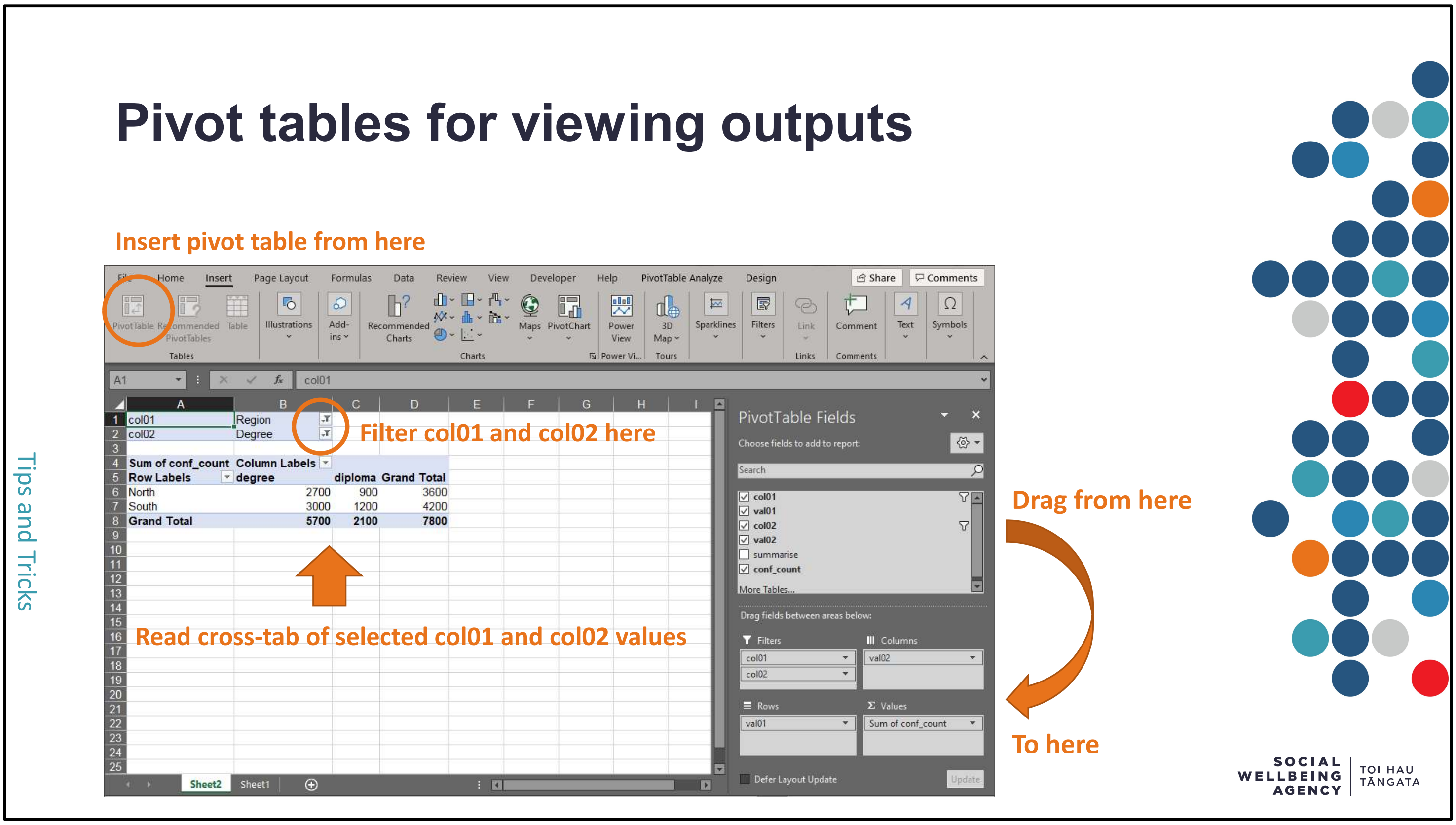
SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

All the tools demonstrated above are designed to work smoothly together. Here we provide an end-to-end demonstration of these tools.

1. Load code for tools into workspace
2. Load a csv file
3. Specify the columns to group by
4. Summarise
5. Confidentialise the summary
6. Write the summary out to a csv file

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 input file and output file
- The names of the columns you want to group by
- The name(s) of the columns you want to summarise
- Whether you want distinct, count, or sum



The long-thin format produced by the summarise and confidentialise tools is well suited for output checking by Stats NZ. However, it is not the best choice of format to inspect or review the results.

We have found Excel pivot tables to be an effective way to present the long-thin table for further analysis.

The example above was produced using the following steps:

- Select the cells containing all the data (including the header row)
- Insert → Pivot Table → Ok
- Drag col01 and col02 to the filters
- Filter col01 to the first measure of interest
- Filter col02 to the second measure of interest
- Drag val01 to the rows and val02 to the columns
- Drag conf_count to the values
- Click on conf_count in the values list → Value Field Settings → Sum → Ok

You can make this type of pivot table interactive by inserting slicers for col01 and col02. Instructions on how to do this can be found online.

Tips and Tricks

Producing entity counts

1) Load

```
rectangular_table = read.csv("./path/input_file_name.csv", as.is = TRUE)

my_cols = c("Qual", "Region", "Age")
all_groups = cross_product_column_names(my_cols, my_cols)
```

2) Create main summary

```
output = summarise_and_label_over_lists(
  rectangular_table, all_groups, list("Identity"),
  make_distinct = FALSE, make_count = TRUE, make_sum = FALSE
)
```

3) Confidentialise

```
conf_output = confidentialise_results(output)
```

4) Create entity counts

```
output_entities = summarise_and_label_over_lists(
  rectangular_table, all_groups, list("Entity_ID"),
  make_distinct = TRUE, make_count = FALSE, make_sum = FALSE
)
```

5) Suppress by entity

```
joined_output = left_join(conf_output, output_entities)
final_output = mutate(joined_output,
  conf_count = ifelse(distinct >= 2, conf_count, NA))
```

6) Write summary to csv

```
write.csv(final_output, "./path/output_file_name.csv")
```

SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

Producing entity counts can be a frustrating task for IDI researchers. Here is one approach to do this effectively using the summarise and confidentialise tools.

The first half is as we have demonstrated before. The part marked in orange is specific to entities. The idea of this part is as follows:

- Use the same groups as for the main output
- Set the summary variable to the ID that defines different identities
- Use `make_distinct = TRUE` to count the number of distinct entities for every grouping
- Join the entity output to the confidentialised output (by default R joins on all columns with the same column names)
- Where the number of distinct entities is at least 2 (the minimum) keep the current count. Otherwise replace it with NA.

We have done entity suppression using a simple `ifelse()` statement. Another option would be to use `apply_small_count_suppression(conf_output, "conf_count ", 2, " distinct ")`

Tips and Tricks

Producing entity counts alternative

1) Load

```
rectangular_table = read.csv("./path/input_file_name.csv", as.is = TRUE)

my_cols = c("Qual", "Region", "Age")
all_groups = cross_product_column_names(my_cols, my_cols)
```

2) Create main summary with entity counts

```
output = summarise_and_label_over_lists(
  rectangular_table, all_groups, list("Entity_ID"),
  make_distinct = TRUE, make_count = TRUE, make_sum = FALSE
)
```

3) Confidentialise

```
conf_output = confidentialise_results(output)
```

4) Suppress by entity

```
final_output = mutate(conf_output,
  conf_count = ifelse(raw_distinct >= 2, conf_count, NA)
)
```

5) Write summary to csv

```
write.csv(final_output, "./path/output_file_name.csv")
```

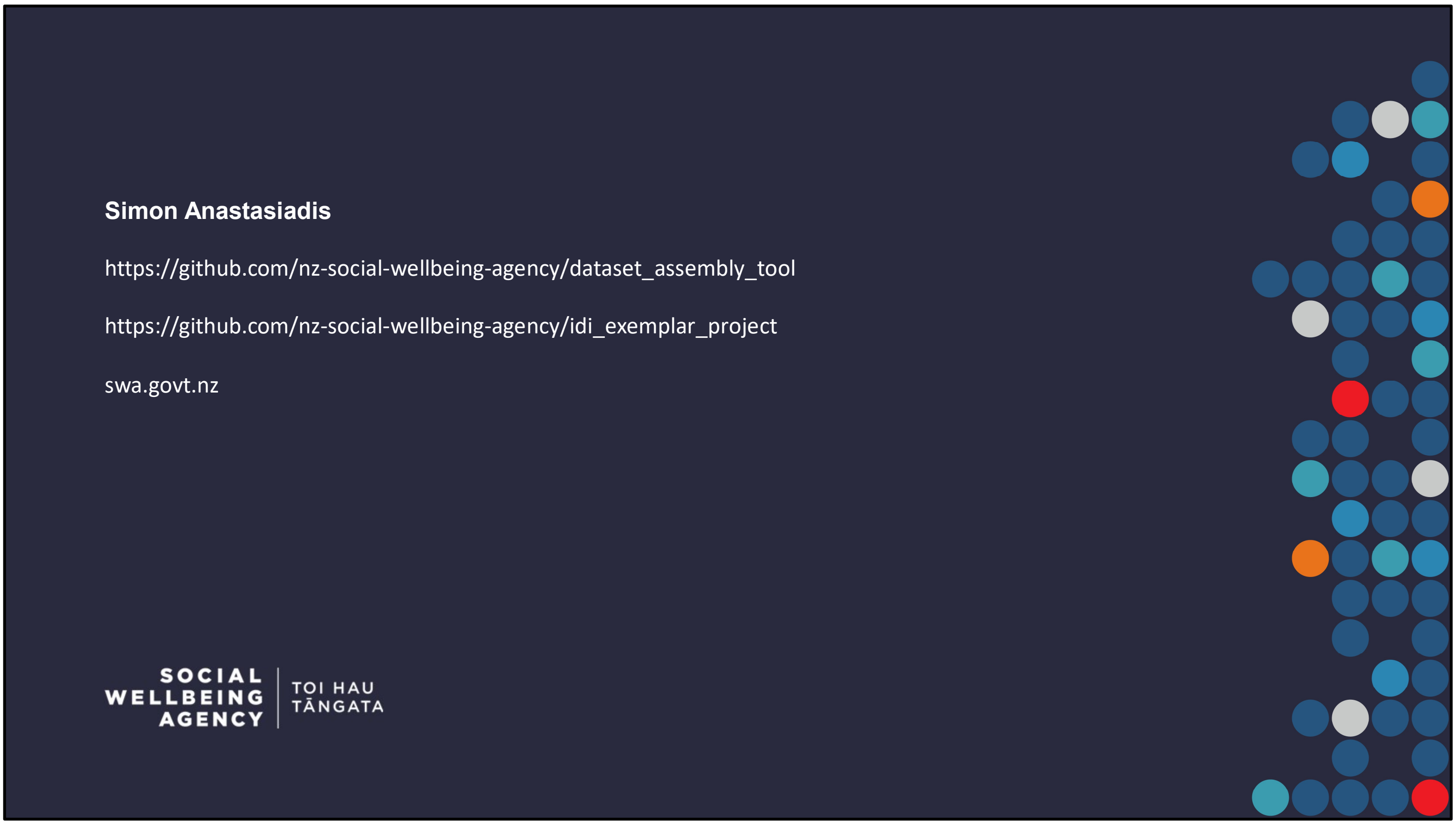
SOCIAL WELLBEING AGENCY | TOI HAU TĀNGATA

For some applications, we can produce entity counts as part of another summary.

Recall that when counting people, we often count the identity column because every person has an ID number. If every person we want to count has an entity ID, then we can use entity ID to count people. This allows us to count both people and entities in the same summary.

This approach differs from the previous slide, as we make a single summary that includes both count and distinct (marked in orange).

- `make_count = TRUE` means the output will contain a column count that contains a count of (non-missing) entity_ID. As everyone has an entity_ID this is equivalent to the number of people.
- `make_distinct = TRUE` means the output will contain a column distinct that contains a count of the number of distinct (non-missing) entity_ID. As each distinct ID represents a different entity, this is the entity count.
- Note that we still need to apply entity count suppression manually. This is not handled automatically by `confidentialise_results`.



Thank you for your time and attention.

The summarise and confidentialise tools demonstrated above have saved significant time for Social Wellbeing Agency staff.
We encourage you to adopt these methods so that you might also benefit from this greater efficiency.