

## ETC5512: Wild Caught Data

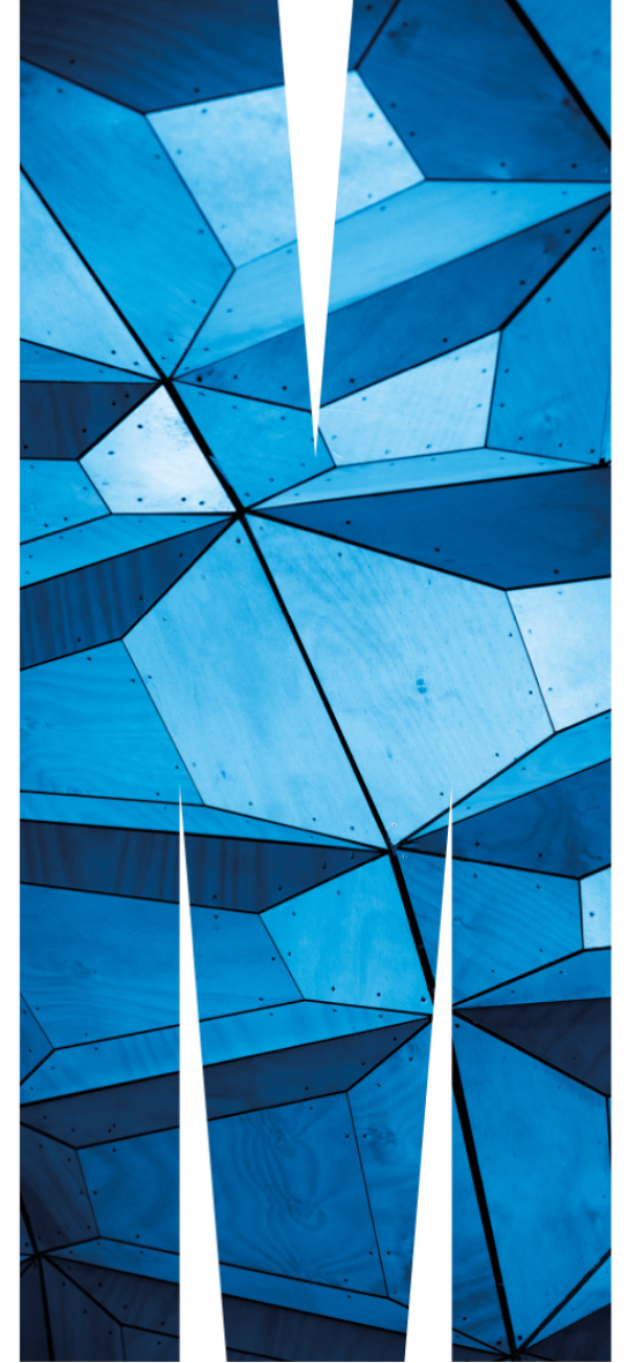
### Australian census

Lecturer: *Emi Tanaka*

Department of Econometrics and Business Statistics

✉ [ETC5512.Clayton-x@monash.edu](mailto:ETC5512.Clayton-x@monash.edu)

📅 Week 4



# Population data

Recall from lecture 2:

“ *Collecting data on the entire population is normally too expensive or infeasible! (If we can, it is called a census.)*

*We therefore collect data only on a subset of the population.*

- There are exceptions to this and one such example, as mentioned, is the **census**.

?

1. When was the last time that the Australian census was run?
2. How often is the census conducted in Australia?
3. Why do we run the census?
4. What data does the Australian census collect?

## Sample survey

- Reduces cost
- Timely collection of data

- Lack of data on sub-population (particularly minorities) or small geographical areas
- Requires careful construction of sampling design
- Estimates are subject to sampling error
- The estimates may not be accurate or reliable
- Estimating and communicating precision of estimates is difficult

## Census

- Data available, even for small geographical areas or subpopulations
- Statistics are not subject to sampling error
- Better accuracy and details

- Expensive or infeasible
- Time consuming to collect all data

Advantages

Disadvantages

# Australian Bureau of Statistics (ABS)

- ABS is the independent statistical agency of the Government of Australia.
- If you are from outside Australia, find the statistical government agency in your country 🛠️, e.g.
  - in 🇯🇵 Japan, this is the [Statistics Bureau of Japan](#),
  - in 🇨🇳 China, the [National Bureau of Statistics of China](#),
  - in 🇮🇳 India, the [Ministry of Statistics and Programme Implementation](#), and
  - in 🇳🇿 New Zealand, the [Statistics New Zealand](#).
- ABS provides key statistics on a wide range of economic, population, environmental and social issues, to assist and encourage informed decision making, research and discussion within governments and the community.





# ABS Census Data

- The first Australian census was held in 1911.
- Since 1961, the census occurs *every 5 years* in Australia.
- The census in 2016 at a *cost of \$440 million*.
- The next census will be held in 2026!
- The ABS is legislated to collect and disseminate census data under the *ABS Act 1975* and *Census and Statistics Act 1905*.
- Similar legislation are in place in many countries.

# Getting the ABS Census Data

 <https://www.abs.gov.au/census/find-census-data>

There are two main types of data that you can download:

- **DataPacks**  <https://datapacks.censusdata.abs.gov.au/datapacks/>
- **GeoPackages**  <https://datapacks.censusdata.abs.gov.au/geopackages/>

# Navigating ABS Census data

- The DataPacks is available only for the 2011 and 2016 census.
- There are slight differences in the available profiles between years, e.g. the General Community Profile in 2016 is a replacement for Basic and Expanded Community Profiles in 2011.
- The information related the census are detailed on the website. See for example [here](#).
- Note: there are sometimes [data corrections](#) at a later date.



Navigating data and deducing what it is often requires you to do some "**detective work**" 🕵️

- Much like real detective work, ***just locating the data and understanding the data variables can take a long time***; the work often is not glamorous; and there's far more attention in "catching criminals" (the discoveries from statistical analysis).



Today,

- We'll navigate through the **personal income data from the 2016 census** together for you to get some "detective" experience
- You'll learn to manipulate strings and a bit about **regular expressions** to deal with string data.
- You'll learn about **tidy data**.



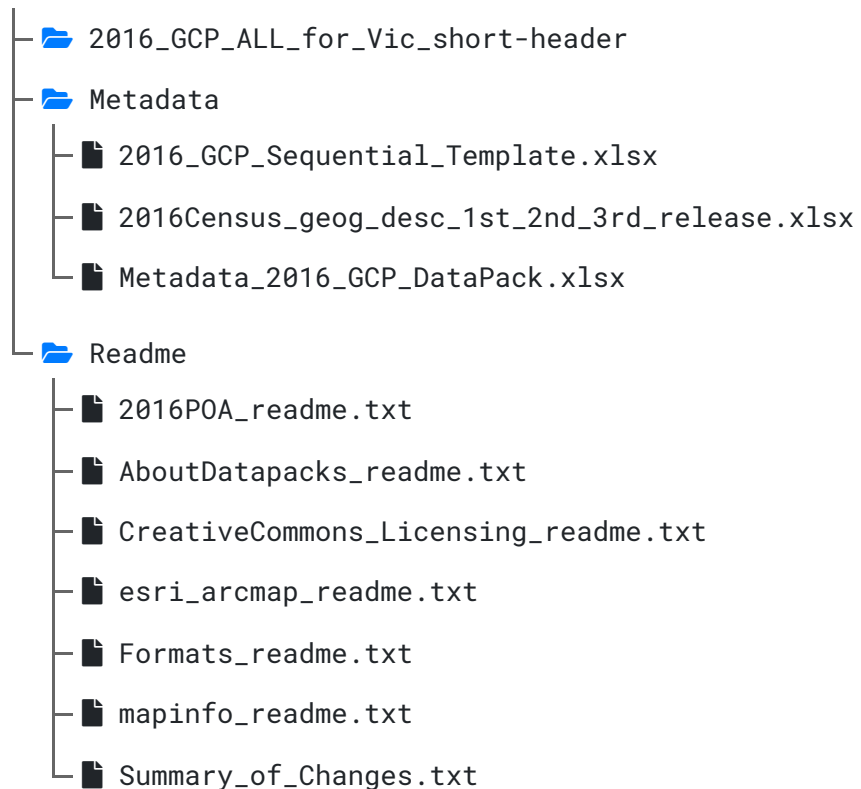
# DataPack directory structure

└─  2016\_GCP\_ALL\_for\_Vic\_short-header

- The data is nested within folders.  
Click on the folder name to see folders and files nested within.
- Preserve the data in the original structure as much as you can! That is, **don't modify the data!**
- Where do we get started??

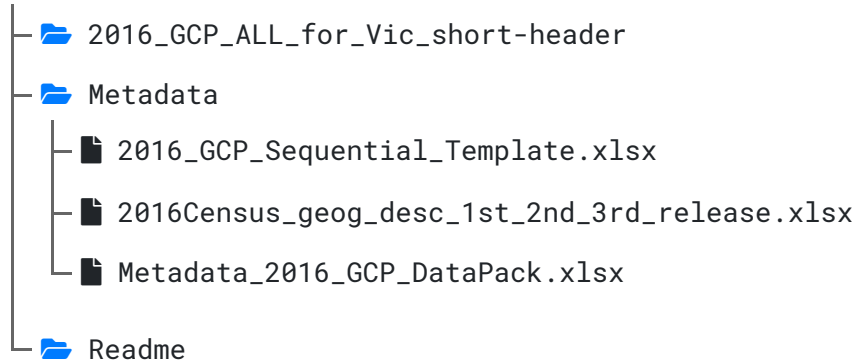
# Getting started

- First, pray hard that there is some description!
- Without some description or understanding of the variables, it will be near impossible to extract meaningful information from the data.



- Readme is a good place to start here (phew!)  
  
"About DataPacks\_readme.md - "Read Me" documentation containing helpful information for users about the data and how it is structured (.md)"
- *But there is no `DataPacks\_readme.md`??*
- We go through other files in the Readme.

# Meta-data

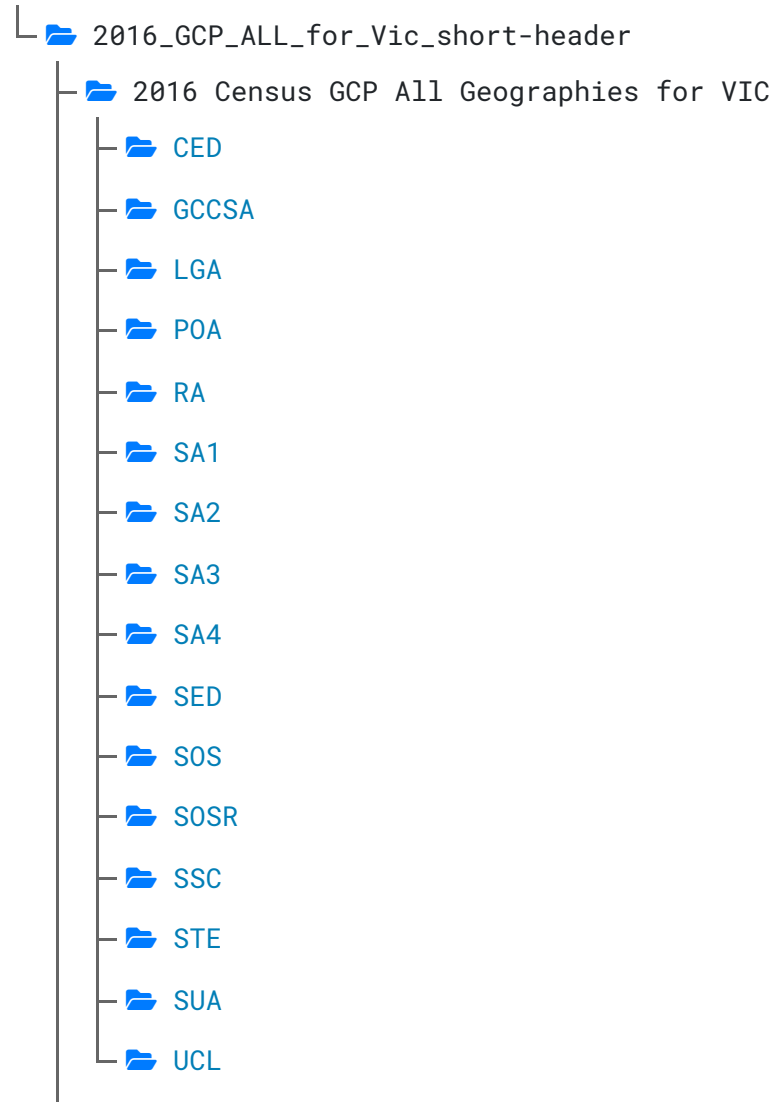


We could also try going through the meta-data.

[Metadata\\_2016\\_GCP\\_DataPack.xlsx](#)

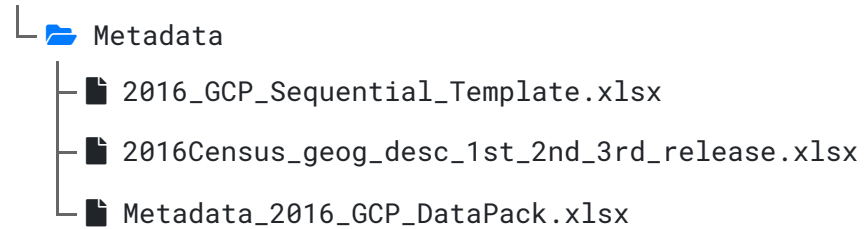
Table number	Table name	Table population
G17	Total Personal Income (Weekly) by Age by Sex	Persons aged 15 years and over
G28	Total Family Income (Weekly) by Family Composition	Families in family households

# Finding Table G17



- Where is Table G17?
- Which Table G17?

# Back to metadata

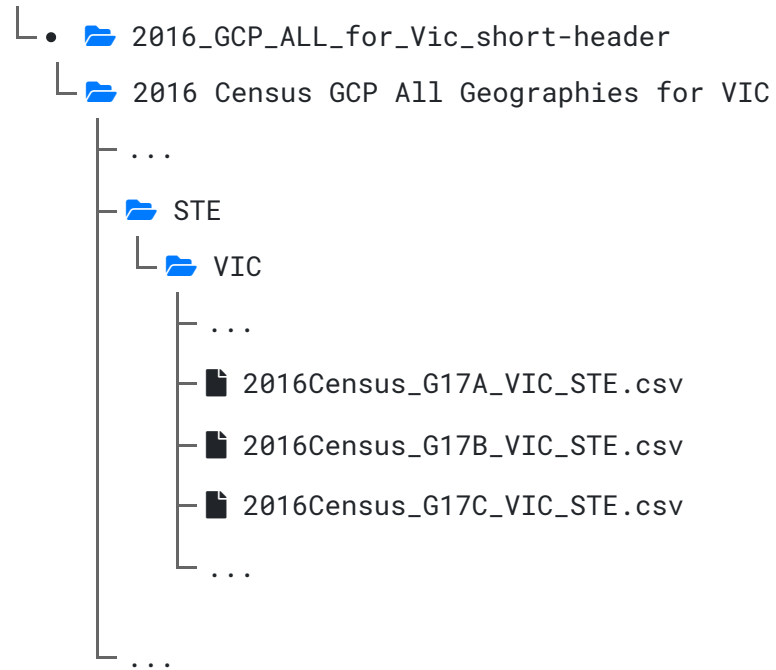


Let's open [2016Census\\_geog\\_desc\\_1st\\_2nd\\_3rd\\_release.xlsx](#)

... and there are the region names of each geographical code.

Let's go with the easy one: [STE Victoria](#).

# Found Table G17?



- G17A, G17B, G17C?

Why is the table organised like this?

# Tables G17A-G17C

2016Census\_G17A\_VIC\_STE.csv

STE_CODE_2016	M_Neg_Nil_income_15_19_yrs	M_Neg_Nil_income_20_24_yrs	M_Neg_Nil_income_25_34_yrs
2	88338	31685	21321

2016Census\_G17B\_VIC\_STE.csv


STE_CODE_2016	F_400_499_15_19_yrs	F_400_499_20_24_yrs	F_400_499_25_34_yrs	F_400_499_35_44_yrs
2	4020	17474	26607	26466

2016Census\_G17C\_VIC\_STE.csv

STE_CODE_2016	P_1000_1249_15_19_yrs	P_1000_1249_20_24_yrs	P_1000_1249_25_34_yrs	P_1000_1249_35_44_yrs
---------------	-----------------------	-----------------------	-----------------------	-----------------------

# Table G17

There are few things to note:

- There are 201 columns in G17A and G17B and 81 columns in G17C.
- Perhaps there is an export limitation for a data that contains more than 200 columns, thus it is broken up into different csv files.
- Which means that you have to join the tables G17A, G17B and G17C as one (you'll do this in the tutorial .



But what does the data show?



# What is Tidy Data?



## Tidy Data Principles

1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell


So what about the ABS 2016 Census Data?

- The table header in fact contains information!
- E.g. `F_400_499_15_19_yrs` is female aged 15-19 years old who earn \$400-499 per week (in Victoria).
- The number in the cells are the **counts**.
- Is the data tidy?

# Tidying the ABS 2016 Census Data

- Ideally we want the data to look like:

age_min	age_max	gender	income_min	income_max	count
15	19	female	400	499	4020

- You can include other information, e.g. geography code (useful if combining with other geographical area) or average age/income.
- Note that some don't have upper bounds, e.g. `M_3000_more_85ov`. In R, `-Inf` and `Inf` are used to represent  $-\infty$  and  $\infty$ , respectively.
- You'll wrangle the data into the tidy form in tutorial 

# Manipulating strings

# Manipulating strings

- The `stringr` package is powered by the `stringi` package which in turn uses the ICU C library to provide fast performance for string manipulation

```
library(tidyverse) # includes `stringr`
```

- Main functions in `stringr` prefix with `str_` (`stringi` prefix with `stri_`) and the **first argument is string** (or a vector of strings)
- What do you think `str_trim` and `str_squish` do?

```
str_trim(c("  Apple ", "  Goji  Berry  "))
```

```
## [1] "Apple"          "Goji  Berry"
```

```
str_squish(c("  Apple ", "  Goji  Berry  "))
```

```
## [1] "Apple"          "Goji Berry"
```

# Base R and `stringr`

Base R	<code>stringr</code>
<code>gregexpr(pattern, x)</code>	<code>str_locate_all(x, pattern)</code>
<code>grep(pattern, x, value = TRUE)</code>	<code>str_subset(x, pattern)</code>
<code>grep(pattern, x)</code>	<code>str_which(x, pattern)</code>
<code>grepl(pattern, x)</code>	<code>str_detect(x, pattern)</code>
<code>gsub(pattern, replacement, x)</code>	<code>str_replace_all(x, pattern, replacement)</code>
<code>nchar(x)</code>	<code>str_length(x)</code>
<code>order(x)</code>	<code>str_order(x)</code>
<code>regexec(pattern, x) + regmatches()</code>	<code>str_match(x, pattern)</code>
<code>regexpr(pattern, x) + regmatches()</code>	<code>str_extract(x, pattern)</code>
<code>regexpr(pattern, x)</code>	<code>str_locate(x, pattern)</code>

Previous

1

2

Next

# Why use **stringr**?

- There are a number of considerations to ensure there is consistency in syntax and user expectation (both for input and output)
- For example, let's consider combining multiple strings into one.

## Base R

## stringr

```
paste0("Area", "1", c("A", "B"))
```

```
## [1] "Area1A" "Area1B"
```

```
paste0("Area", "1", c("A", NA, "C"))
```

```
## [1] "Area1A" "Area1NA" "Area1C"
```

```
str_c("Area", "1", c("A", "B"))
```

```
## [1] "Area1A" "Area1B"
```

```
str_c("Area", "1", c("A", NA, "C"))
```

```
## [1] "Area1A" NA "Area1C"
```

- If the Base R result is preferable then NA can be replaced with character with `str_replace_na("A", NA, "C")` first

## Case study Aussie Local Government Area

```
LGA <- ozmaps::abs_lga %>% pull(NAME)
```

```
LGA[1:7]
```

```
## [1] "Broken Hill (C)" "Waroona (S)"      "Toowoomba (R)"    "West Arthur (S)"
## [5] "Moreton Bay (R)" "Etheridge (S)"    "Cleve (DC)"
```

C = Cities

A = Areas

RC = Rural Cities

B = Boroughs

S = Shires

DC = District Councils

M = Municipalities

T = Towns

AC = Aboriginal Councils

RegC = Regional Councils

 **Extract the LGA status from the LGA names**

How?

# Extracting the string

```
str_extract(LGA, "\\(.+\\)")
```

```
## [1] "(C)" "(S)" "(R)" "(S)" "(R)"
## [6] "(S)" "(DC)" "(DC)" "(DC)" "(DC)"
## [11] "(DC)"
## [16] "(A)"
## [21] "(A)"
## [26] "(DC)"
## [31] "(S)"
## [36] "(R)"
## [41] "(S)"
## [46] "(AC)"
## [51] "(A)"
## [56] "(S)"
## [61] "(C)"
## [66] "(C)"
## [71] "(R)" "(R)" "(S)" "(B)" "(DC)"
```



- What is "\\(.+\\)"???
- This is a pattern expressed as **regular expression** or **regex** for short
- Note in R, you have to add an extra `\` when `\` is included in the **pattern** (yes this means that you can have a lot of backslashes... just keep adding `\` until it works! Enjoy [this xkcd comic](#).)
- From R v4.0.0 onwards, you can use raw string to eliminate all the extra `\`, e.g. `r"(\(.+\))"` is the same as "\\(.+\\)"



# Regular expressions Part 1

- **Regular expression**, or **regex**, is a string of characters that define a search pattern for text
- Regular expression is... hard, but comes up often enough that it's worth learning

```
ozanimals <- c("koala", "kangaroo", "kookaburra", "numbat")
```

## = Basic match

```
str_detect(ozanimals, "oo")
```

```
## [1] FALSE TRUE TRUE FALSE
```

```
str_extract(ozanimals, "oo")
```

```
## [1] NA "oo" "oo" NA
```

```
str_match(ozanimals, "oo")
```

```
##      [,1]
```

```
## [1,] NA
```

```
## [2,] "oo"
```

```
## [3,] "oo"
```

```
## [4,] NA
```

# Regular expressions Part 2

## Meta-characters

- "." a wildcard to match any character except a new line

```
str_starts(c("color", "colour", "colour", "red-column"), "col...")  
## [1] FALSE TRUE TRUE FALSE
```

- "(.|.)" a marked subexpression with alternate possibilities marked with |

```
str_replace(c("love", "move", "stove", "drove"), "(l|dr|st)o", "ha")  
## [1] "have" "move" "have" "have"
```

- "[...]" matches a single character contained in the bracket

```
str_replace_all(c("cake", "cookie", "lamington"), "[aeiou]", "_")  
## [1] "c_k_" "c__k__" "l_m_ngt_n"
```

# Regular expressions Part 3

## Meta-character quantifiers

- "?" zero or one occurrence of preceding element

```
str_extract(c("color", "colour", "colour", "red"), "colou?r")
```

```
## [1] "color" NA "colour" NA
```

- "\*" zero or more occurrence of preceding element

```
str_extract(c("color", "colour", "colour", "red"), "colou*r")
```

```
## [1] "color" "colour" "colour" NA
```

- "+" one or more occurrence of preceding element

```
str_extract(c("color", "colour", "colour", "red"), "colou+r")
```

```
## [1] NA "colour" "colour" NA
```

# Regular expressions Part 4

- "`{n}`" preceding element is matched exactly `n` times

```
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){2}", "-")
```

```
## [1] "-"      "-na"    "bana"   "-nana"
```

- "`{min, }`" preceding element is matched `min` times or more

```
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){2,}", "-")
```

```
## [1] "-"      "-"      "bana"   "-"
```

- "`{min,max}`" preceding element is matched at least `min` times but no more than `max` times

```
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){1,2}", "-")
```

```
## [1] "-"      "-na"    "-"      "-nana"
```

# Regular expressions Part 5

## = Character classes

- `[ :alpha: ]` or `[ A-Za-z ]` to match alphabetic characters
- `[ :alnum: ]` or `[ A-Za-z0-9 ]` to match alphanumeric characters
- `[ :digit: ]` or `[ 0-9 ]` or `\\d` to match a digit
- `[ ^0-9 ]` to match non-digits
- `[ a-c ]` to match a, b or c
- `[ A-Z ]` to match uppercase letters
- `[ a-z ]` to match lowercase letters
- `[ :space: ]` or `[ \\t\\r\\n\\v\\f ]` to match whitespace characters
- and more...

# View matches with regular expressions

```
str_view(c("banana", "bananana", "bana", "banabanana"), "ba(na){1,2}")
```

```
banana
```

```
bananana
```

```
bana
```

```
banabanana
```



- When a function in `stringr` ends with `_all`, all matches of the pattern are considered
- The one *without* `_all` only considers the first match

```
str_view_all(c("banana", "bananana", "bana", "banabanana"), "ba(na){1,2}")
```

```
banana
```

```
bananana
```

```
bana
```

```
banabanana
```

## Back to **Extracting the string**

```
str_extract(LGA, "\\(.+\\)") %>%  
  table()
```

```
## .  
##      (A)      (AC)      (B)      (C) (C) (NSW)      (C) (SA) (C) (Vic.)  
##      100      2      1      120      2      1      2  
##      (DC) (DC) (SA)      (M) (M) (Tas.)      (R) (R) (Qld)      (RC)  
##      40      1      23      4      38      1      7  
##      (RegC)      (S) (S) (Qld)      (T)  
##      1      182      1      12
```

Where the same Local Government Area name appears in different States or Territories, the State or Territory abbreviation appears in parenthesis after the name. Local Government Area names are therefore unique.

-Australian Bureau of Statistics

## Retry **Extracting the string**

```
str_extract(LGA, "\\([^\)]+\\)") %>%  
  # remove the brackets  
  str_replace_all("\\([\\])", "") %>%  
  table()
```

```
## .  
##      A    AC    B     C    DC     M     R    RC  RegC     S     T  
## 100     2     1  125   41   27    39    7     1   183   12
```

- "[ ]" for single character match
- We want to match ( and ) but these are meta-characters
- So we need to escape it to have it as a literal: \  
( and \  
)
- But we must escape the escape character... so it's actually \  
\  
( \  
\  
)



## R v4.0.0 Extracting the string

```
str_extract(LGA, r"(\([^)]+)\)") %>%  
  # remove the brackets  
  str_replace_all(r"([\(\)])", "") %>%  
  table()
```

```
## .  
##   A   AC   B   C   DC   M   R   RC  RegC   S   T  
## 100   2   1 125  41  27  39   7   1 183 12
```

- If using R v4.0.0 onwards, you can use the raw string version instead

**Back to Census**

# Raw Data vs. Aggregated Data

- Although the data collected was from individual households surveying each person in the household (see sample form [here](#)), the downloaded data are **aggregated**.
- Aggregated data presents summary statistics from the *raw data*. When the only summary statistics are counts then it is generally called *frequency data*.
- The raw data collected would be similar to the form

household_id	person	gender	age	marital_status	income_per_week
1	John Smith	F	40	Married	400-499
1	Jane Smith	M	39	Married	300-399
1	David Smith	M	10	Never married	Nil
1	Mary Smith	F	8	Never married	Nil
2	John Citizen	M	32	Never married	400-499
2	Jane Citizen	F	33	Never married	1750-1999

# What you lose in aggregate data

- For aggregate data, there are less scope for you to draw insights conditioned on other variables.
- E.g. based on frequency data alone, you cannot answer questions like: how many middle income families with 2 children?
- Raw data are desirable if you can get hold of it!

## Trust and skepticism

- By the way, did you notice anything odd about the dummy data presented in the last slide?
- John Smith was recorded as female and Jane Smith as male. Data may have been incorrectly recorded.
- How much do you trust the aggregate data?
- Have some healthy dose of skepticism in your data.

# Data Confidentiality

- The data is not just aggregated, but it is also **anonymised**
- E.g. in [2016\\_GCP\\_Sequential\\_Template.xlsx](#), Sheet "G 17a", footnote says "*Please note that there are **small random adjustments** made to all cell values to protect the confidentiality of data. These adjustments may cause the sum of rows or columns to differ by small amounts from table totals.*"



Do you think that you'll get the same numbers if you use the ones from different geographical code? E.g. [SA1](#) and [STE](#).

- You can check this in the tutorial 



## Summary

- We went through how to locate and understand the data variables for the personal income data from the 2016 Australian census.
- We know some limitations with this data.
- We learnt how to manipulate strings and a little about regular expression.
- We learnt about what tidy data is.



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Lecturer: *Emi Tanaka*

Department of Econometrics and Business Statistics

✉ [ETC5512.Clayton-x@monash.edu](mailto:ETC5512.Clayton-x@monash.edu)

📅 Week 4

