MONASH University MALAYSIA

MONASH BUSINESS

Tidy Data with tidyr

ETW2001 Foundations of Data Analysis and Modelling

Manjeevan Singh Seera

Accredited by: AACSB ACCREDITED

Advanced Signatory: PRME an initiative of the United Nations Global Compact

Outline

- ❑ **Tidy Data with tidyr**
  - ❑ Tidy Data
  - ❑ Pivoting
  - ❑ Separating and Uniting
  - ❑ Case Study

Introduction

In this slides, we will learn a consistent way to organize data in R, called tidy data.

Getting your data into this format requires some **upfront work**, but that work **pays off** in the long term. Once you have tidy data and the tidy tools provided by packages in the tidyverse, you will spend much less time munging data from one representation to another, allowing you to spend more time on the analytic questions at hand.

Source: https://r4ds.had.co.nz/tidy-data.html

Prerequisites

The focus on tidyr, a package that provides a bunch of tools to help tidy up your messy datasets. It is a member of the core tidyverse.

```
library(tidyverse)
```

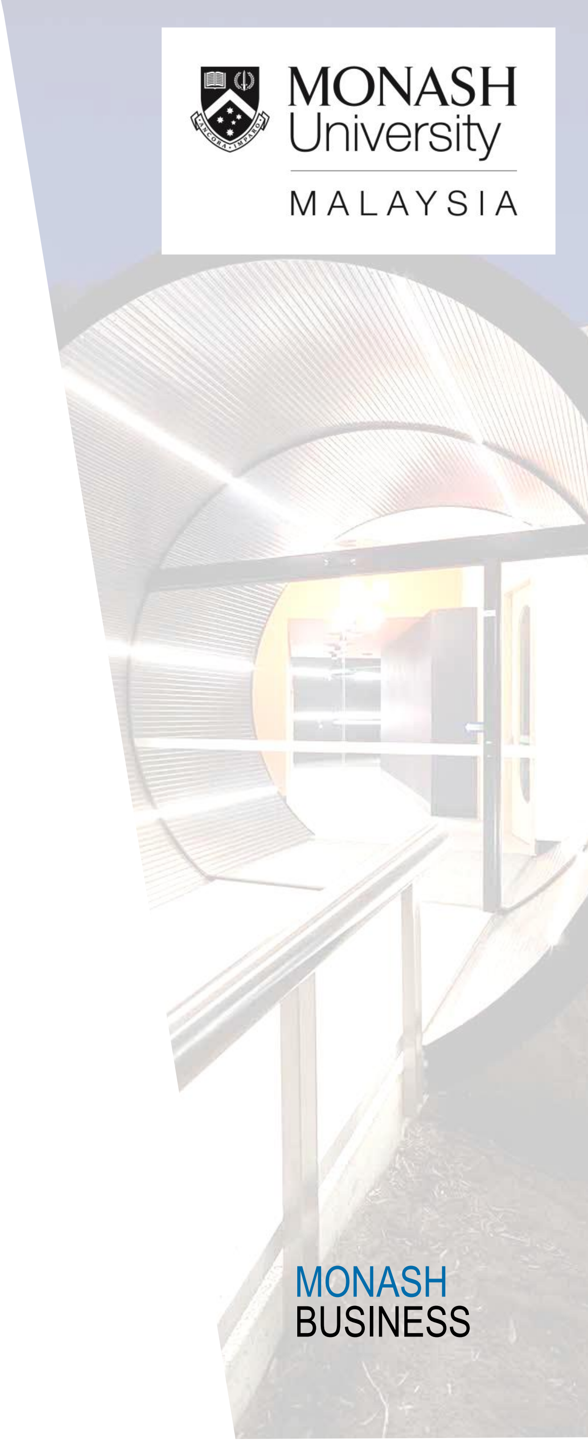
```
── Attaching packages ───────────────

✓ ggplot2 3.3.2     ✓ purrr   0.3.4
✓ tibble  3.0.4     ✓ dplyr   1.0.2
✓ tidyr   1.1.2     ✓ stringr 1.4.0
✓ readr   1.4.0     ✓ forcats 0.5.0
```

Outline

✓ Tidy Data with tidyr

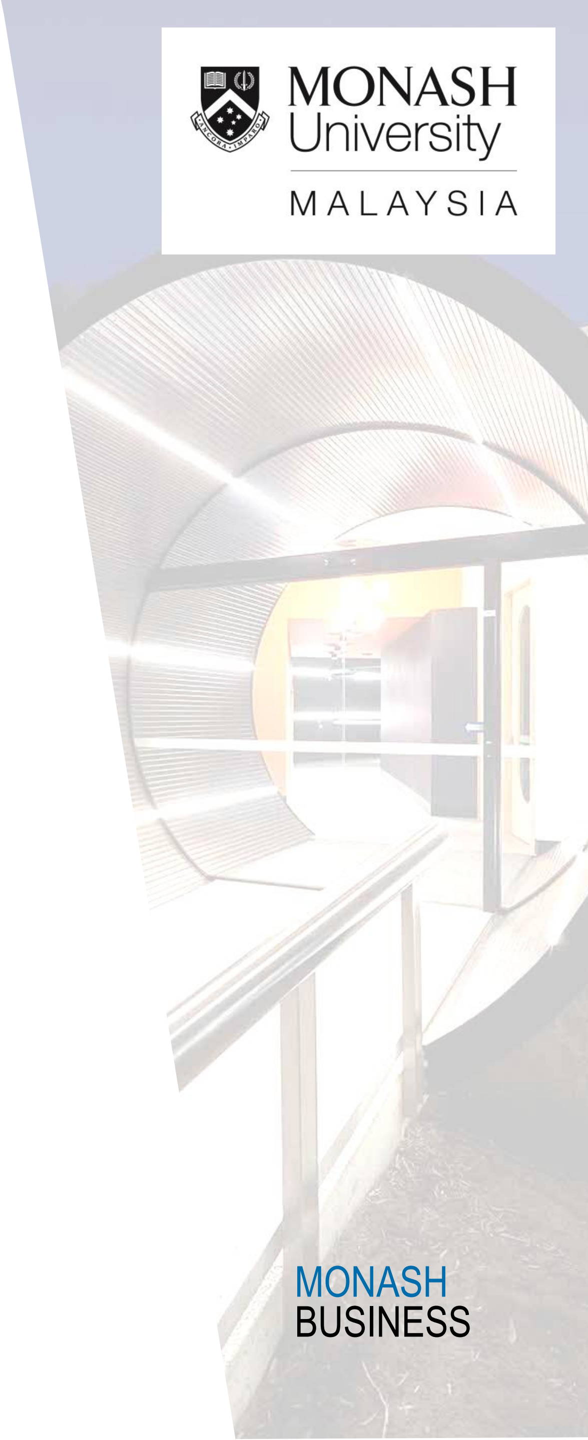
- ❑ **Tidy Data**
- ❑ Pivoting
- ❑ Separating and Uniting
- ❑ Case Study



Dataset

The first dataset used is a subset of the data contained in the World Health Organization Global Tuberculosis Report.

table1, table2, table3, table4a, table4b, and table5 all display the number of TB cases documented by the World Health Organization in Afghanistan, Brazil, and China between 1999 and 2000. The data contains values associated with four variables (country, year, cases, and population), but each table organizes the values in a different layout.



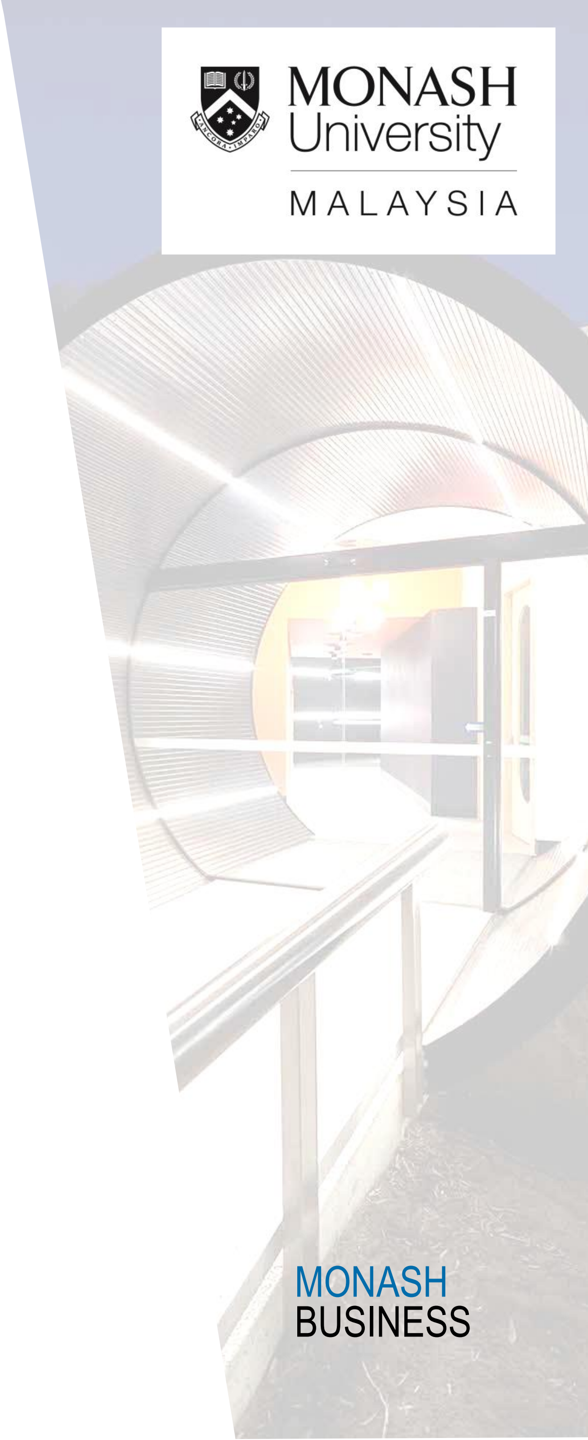
Tidy data

The same underlying data can be represented in **multiple ways**. Example below shows the same data organized, with four variables: country, year, population, and cases. However, each dataset organizes the values in a different way.

```
table1
```

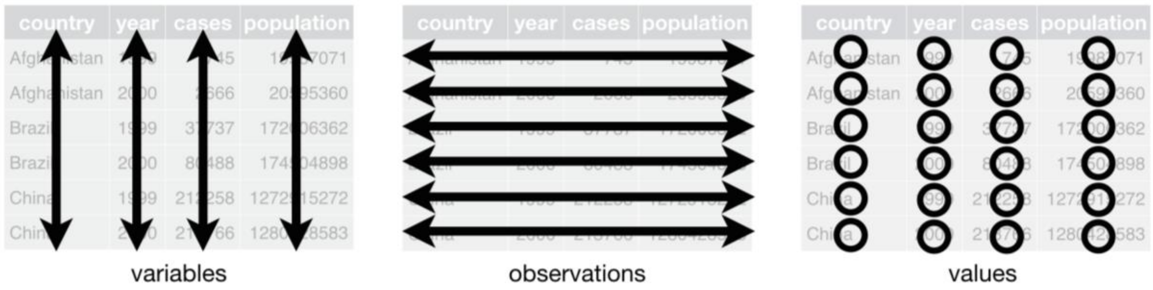
A tibble: 6 × 4

country	year	cases	population
<chr>	<int>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



Tidy data

There are 3 interrelated rules which make a dataset tidy:
- Each variable must have its own **column**.
- Each observation must have its own **row**.
- Each value must have its own **cell**.



Following 3 rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

Tidy data

These **3** rules are interrelated because it's impossible to only satisfy **2** of the **3**.
That interrelationship leads to an even simpler set of practical instructions:

- Put each dataset in a tibble.
- Put each variable in a column.

In this example, only table1 is tidy. It's the only representation where each column is a variable.

MONASH
BUSINESS

Tidy data

Why ensure that your data is tidy? Two main advantages:

- General advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.
- Specific advantage to placing variables in columns because it allows R's vectorised nature to shine. Most built-in R functions work with vectors of values, which makes transforming tidy data feel particularly natural.

dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data.

Outline

- ✓ Tidy Data with tidyr
  - ✓ Tidy Data
  - ❑ Pivoting
  - ❑ Separating and Uniting
  - ❑ Case Study

Pivoting

Most data that you will encounter will be untidy. Data is often organized to facilitate some use other than analysis. As an example, data is often organized to make entry as easy as possible.

For most analysis, we will need to do tidying. First, we'll need to figure out what the variables and observations are. Sometimes this is easy; other times there might be a need to consult with the people who originally generated the data. Next step is to resolve one of two common problems:

- One variable might be spread across multiple columns,
- One observation might be scattered across multiple rows.

Generally, a dataset only has one of the 2 problems. To fix this issues, we'll look at important functions in tidyr: `pivot_longer()` and `pivot_wider()`.

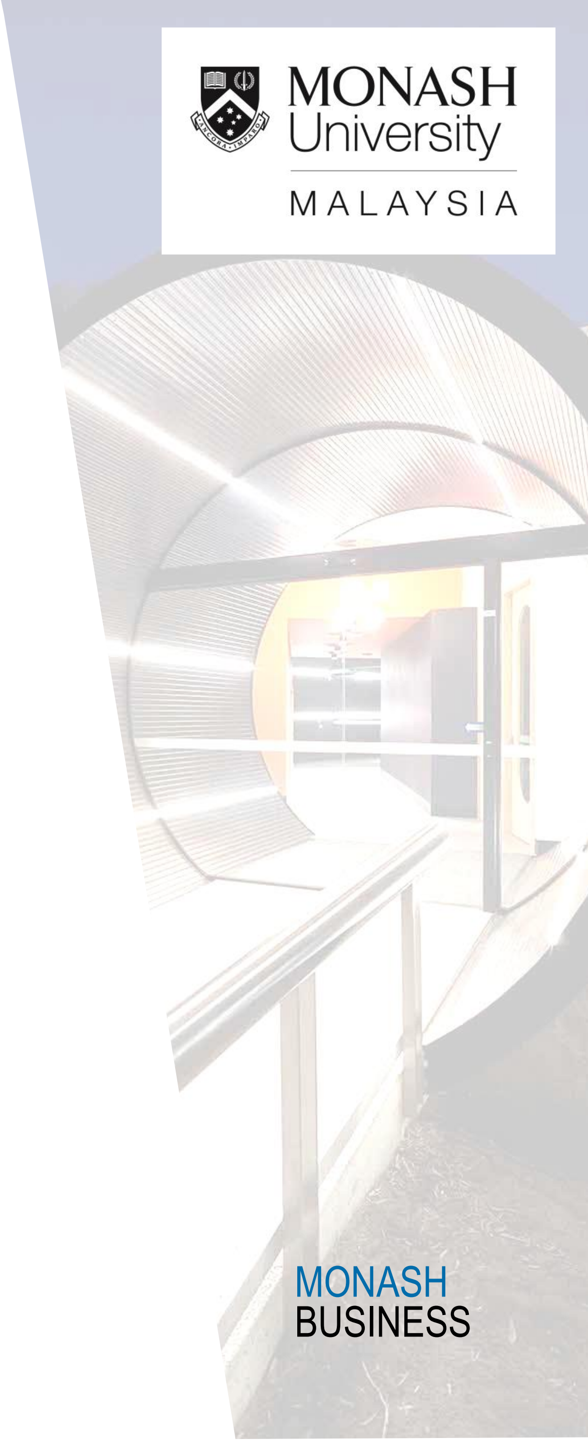
Longer

A typical problem is a dataset where some of the column names are not names of variables, but values of a variable. For example, table4a: the column names 1999 and 2000 represent values of the year variable, the values in the 1999 and 2000 columns represent values of the cases variable, and each row represents two observations, not one.

```
table4a
```

A tibble: 3 × 3

	country	1999	2000
	<chr>	<int>	<int>
1	Afghanistan	745	2666
2	Brazil	37737	80488
3	China	212258	213766



Longer

MONASH University MALAYSIA

To tidy a dataset like this, we need to **pivot** the offending columns into a new pair of variables. For this operation, we need **3** parameters:

- Set of columns whose names are values, not variables (for example the columns 1999 and 2000).
- Name of the variable to move the column names to (example is year).
- Name of the variable to move the column values to (example is cases).

Together those parameters generate the call to `pivot_longer()`:

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

A tibble: 6 × 3

country	year	cases
<chr>	<chr>	<int>
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

Pivot

Columns to pivot are specified with `dplyr::select()` style notation. For “1999” and “2000”, these are non-syntactic names (as they don’t start with a letter) so we have to surround them in backticks.

year and cases do not exist in table4a so we put their names in quotes.

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

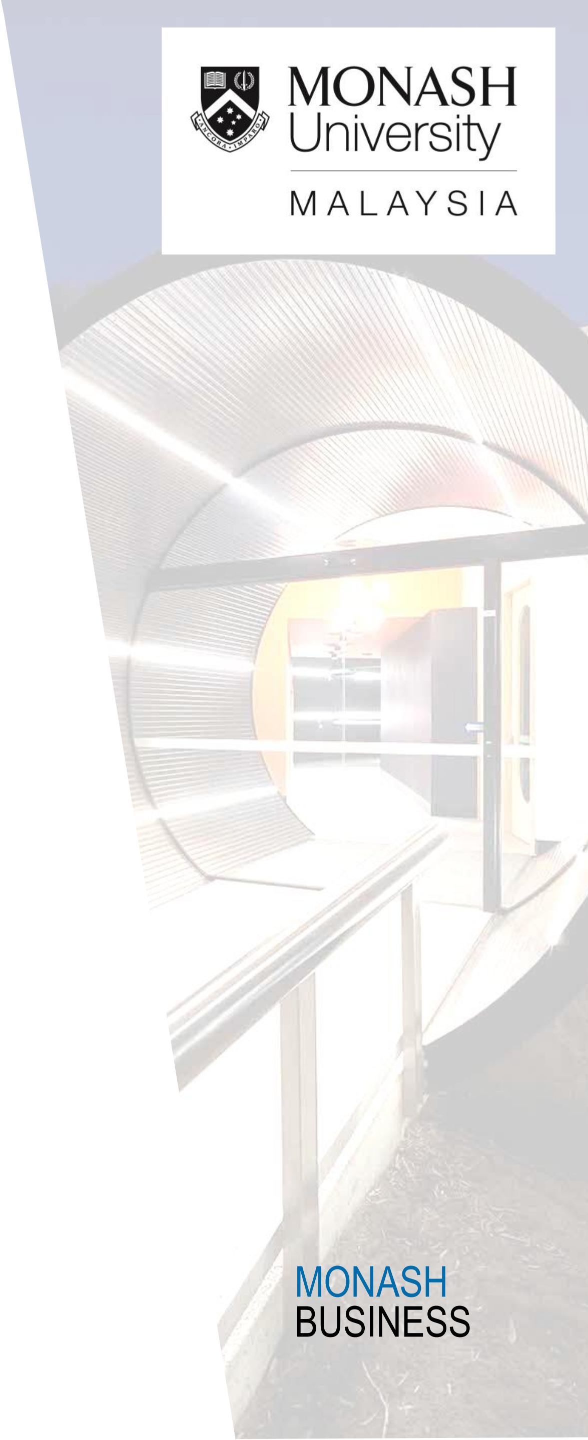
Fig. 1: Pivoting table4 into a longer, tidy form

Longer

In the final result, the **pivoted columns** are dropped, and we get new year and cases columns. Else, relationships between the original variables are preserved (as shown in Fig. 1).

`pivot_longer()` makes datasets:

- longer by increasing the number of rows,
- decreasing the number of columns.



Wider

MONASH University MALAYSIA

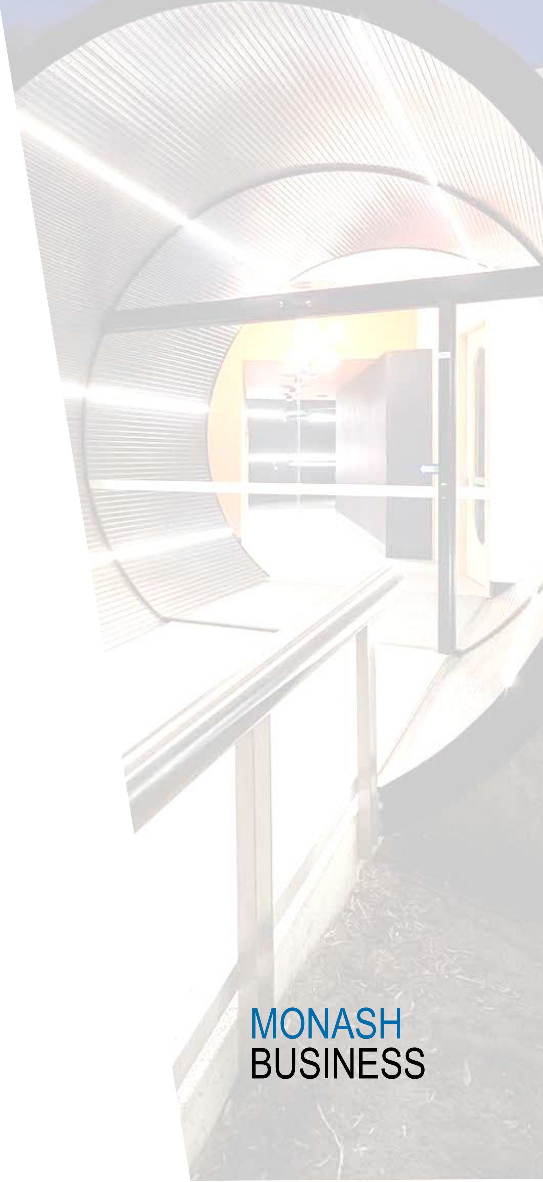
`pivot_wider()` is the **opposite** of `pivot_longer()`.

Use it when an observation is scattered across multiple rows. As an example, take table2: an observation is a country in a year, but each observation is spread across two rows.

```
table2
```

A tibble: 12 × 4

country	year	type	count
<chr>	<int>	<chr>	<int>
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583



MONASH BUSINESS

Wider

To tidy this up, we first analyze the representation in similar way to `pivot_longer()`. Only **2** parameters are needed:

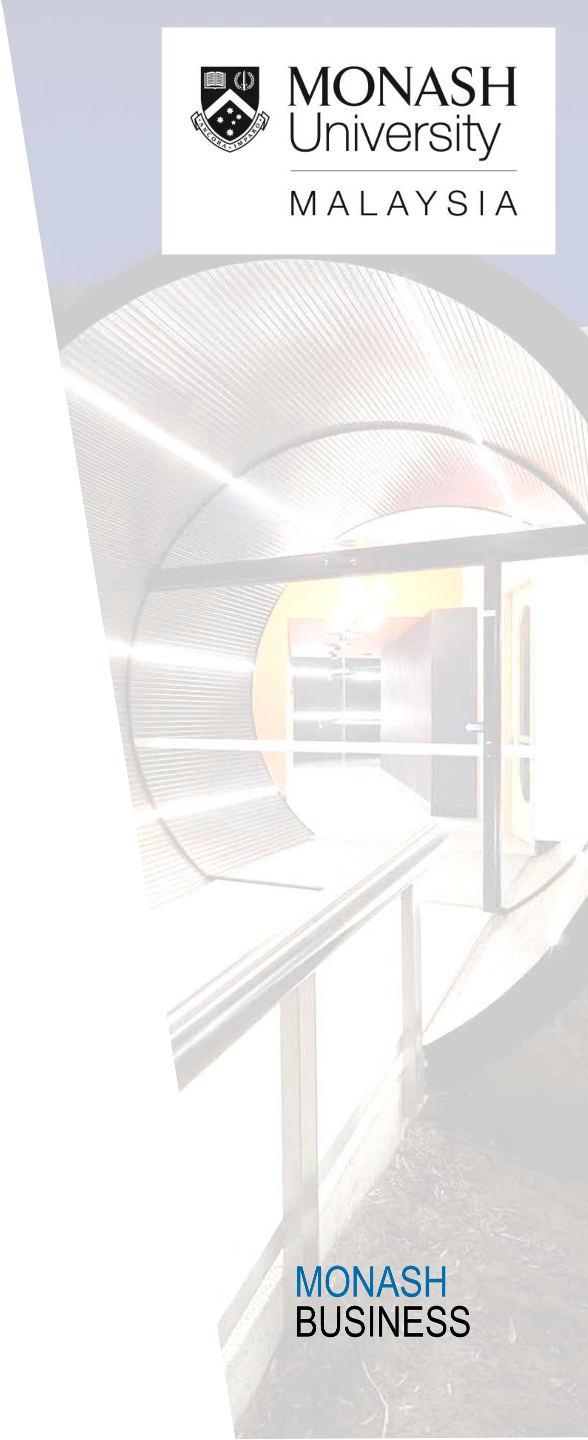
- Column to take variable names from (type).
- Column to take values from (count).

Once we've figured that out, we can use `pivot_wider()` shown below and visually in Fig. 2.

```
table2 %>%
    pivot_wider(names_from = type, values_from = count)
```

A tibble: 6 × 4

country	year	cases	population
<chr>	<int>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



Wider and longer

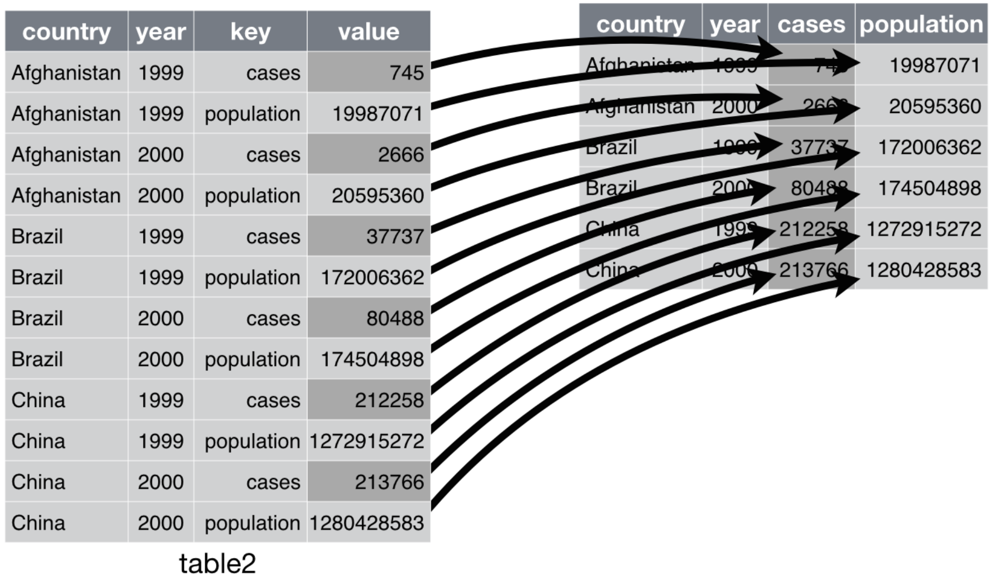


Fig. 2: Pivoting table2 into a “wider”, tidy form.

Both `pivot_wider()` and `pivot_longer()` are complements, where

- `pivot_longer()` makes wide tables narrower and longer,
- `pivot_wider()` makes long tables shorter and wider.

Outline

- ✓ Tidy Data with tidyr
  - ✓ Tidy Data
  - ✓ Pivoting
  - ❑ **Separating and Uniting**
  - ❑ Case Study

Separating and uniting

In table3, there is a different problem: there is 1 column (**rate**) that contains two variables (cases and population).

To fix this problem, we'll need the `separate()` function.

- We'll learn about complement of `separate()`: `unite()`, which you use if a single variable is spread across multiple columns.

Separate

`separate()` pulls apart **1** column into **multiple** columns, by splitting wherever a **separator** character appears.

```
table3
```

A tibble: 6 × 3

	country	year	rate
	<chr>	<int>	<chr>
1	Afghanistan	1999	745/19987071
2	Afghanistan	2000	2666/20595360
3	Brazil	1999	37737/172006362
4	Brazil	2000	80488/174504898
5	China	1999	212258/1272915272
6	China	2000	213766/1280428583

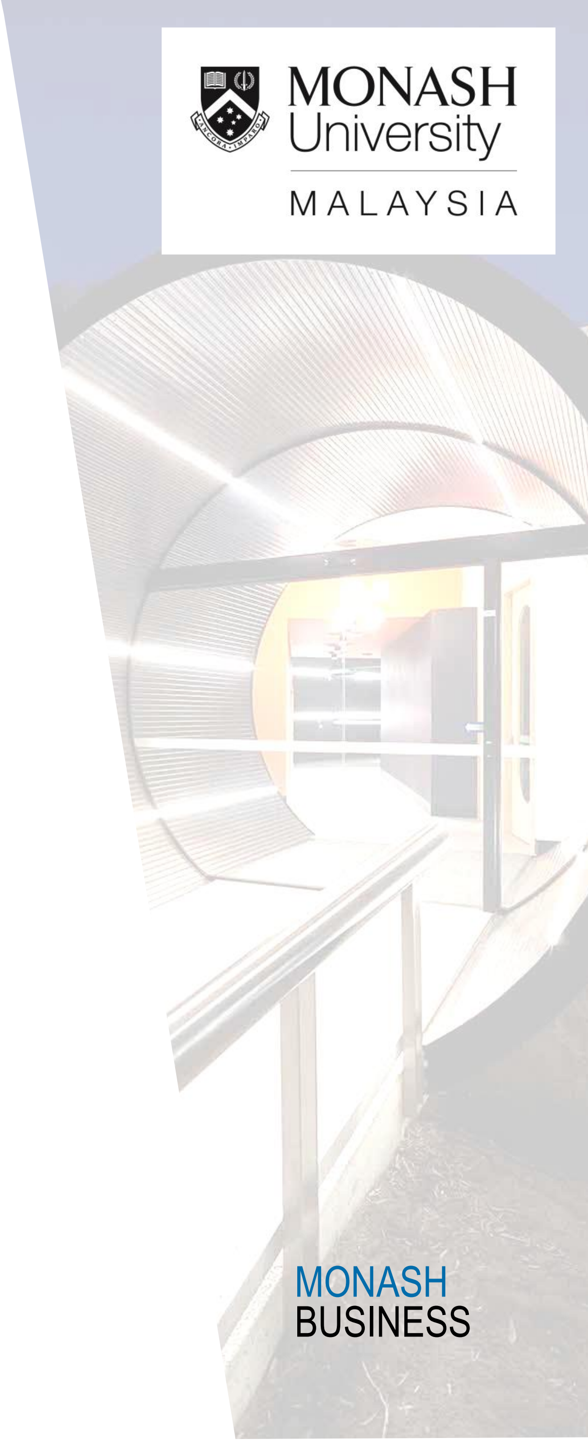
Separate

The rate column contains both **cases** and **population** variables, and we need to split it into two variables. `separate()` takes the name of the column to separate, and the names of the columns to separate into, as shown in Fig 3.

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

A tibble: 6 × 4

country	year	cases	population
<chr>	<int>	<chr>	<chr>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



Separate

country	year	rate
Afghanistan	1999	**745** / 19987071
Afghanistan	2000	**2666** / 20595360
Brazil	1999	**37737** / 172006362
Brazil	2000	**80488** / 174504898
China	1999	**212258** / 1272915272
China	2000	**213766** / 1280428583

table3

country	year	cases	population
Afghanistan	1999	**745**	19987071
Afghanistan	2000	**2666**	20595360
Brazil	1999	**37737**	172006362
Brazil	2000	**80488**	174504898
China	1999	**212258**	1272915272
China	2000	**213766**	1280428583

Fig. 3: Separating table3 makes it tidy

MONASH University MALAYSIA

MONASH BUSINESS

Separate

`separate()` splits values wherever it sees a non-alphanumeric character (character that isn't a number or letter). As an example, `separate()` was used split the values of rate at the forward slash characters. Should we wish to use a specific character to separate a column, we can pass the **character** (eg. /) to the **sep** argument of `separate()`. Code can be rewritten as follows.

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

A tibble: 6 × 4

country	year	cases	population
<chr>	<int>	<chr>	<chr>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

MONASH University MALAYSIA

MONASH BUSINESS

Separate

Look at the column types: notice that cases and population are **character (chr)** columns. This is the **default** behavior in `separate()`: it leaves the type of the column as is. It's not very useful as those really are numbers. Ask `separate()` to try and convert to better types using **convert = TRUE**.

```
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

A tibble: 6 × 4

country	year	cases	population
<chr>	<int>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Separate

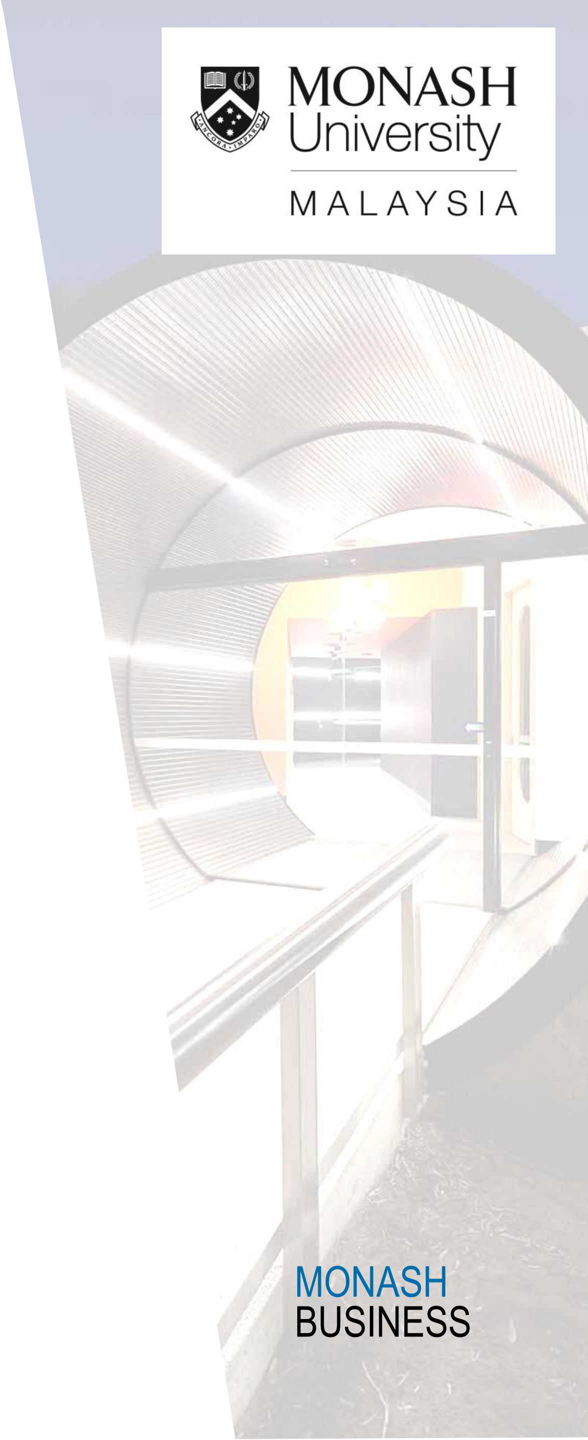
We can pass a vector of integers to sep. `separate()` will interpret integers as positions to split at. Positive values start at 1 on the far-left of the strings; negative value start at -1 on the far-right of the strings. When using integers to separate strings, the length of sep should be one less than the number of names in into.

Let's separate last **2** digits of each year.

```
table3 %>%
  separate(year, into = c("century", "year"), sep = 2)
```

A tibble: 6 × 4

country	century	year	rate
<chr>	<chr>	<chr>	<chr>
Afghanistan	19	99	745/19987071
Afghanistan	20	00	2666/20595360
Brazil	19	99	37737/172006362
Brazil	20	00	80488/174504898
China	19	99	212258/1272915272
China	20	00	213766/1280428583



Unite

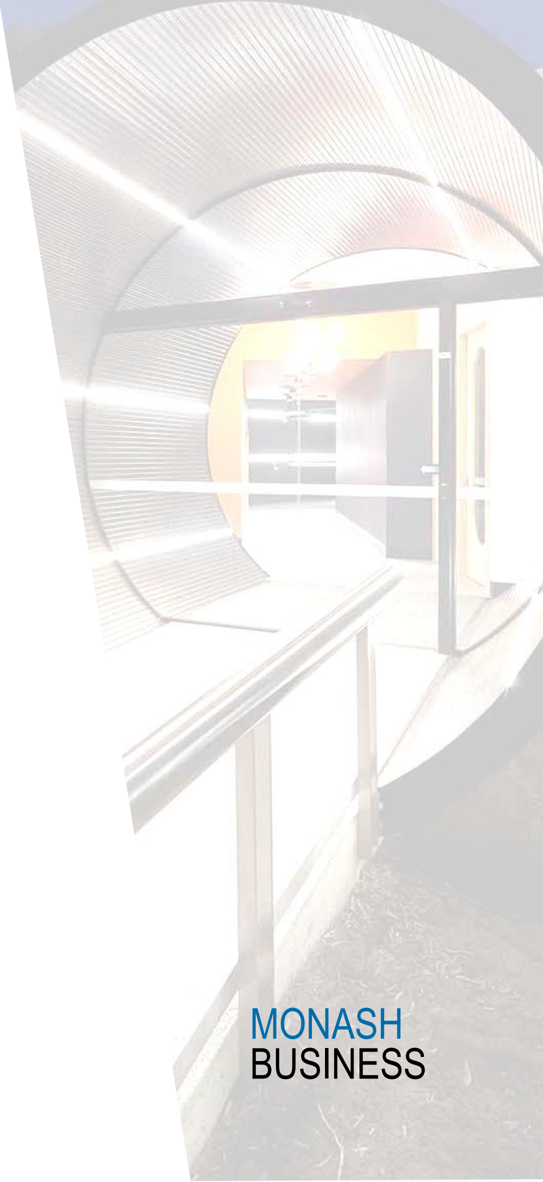
`unite()` is the **inverse** of `separate()`: it combines multiple columns into a single column.

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

table6

Fig. 4: Uniting table5 makes it tidy



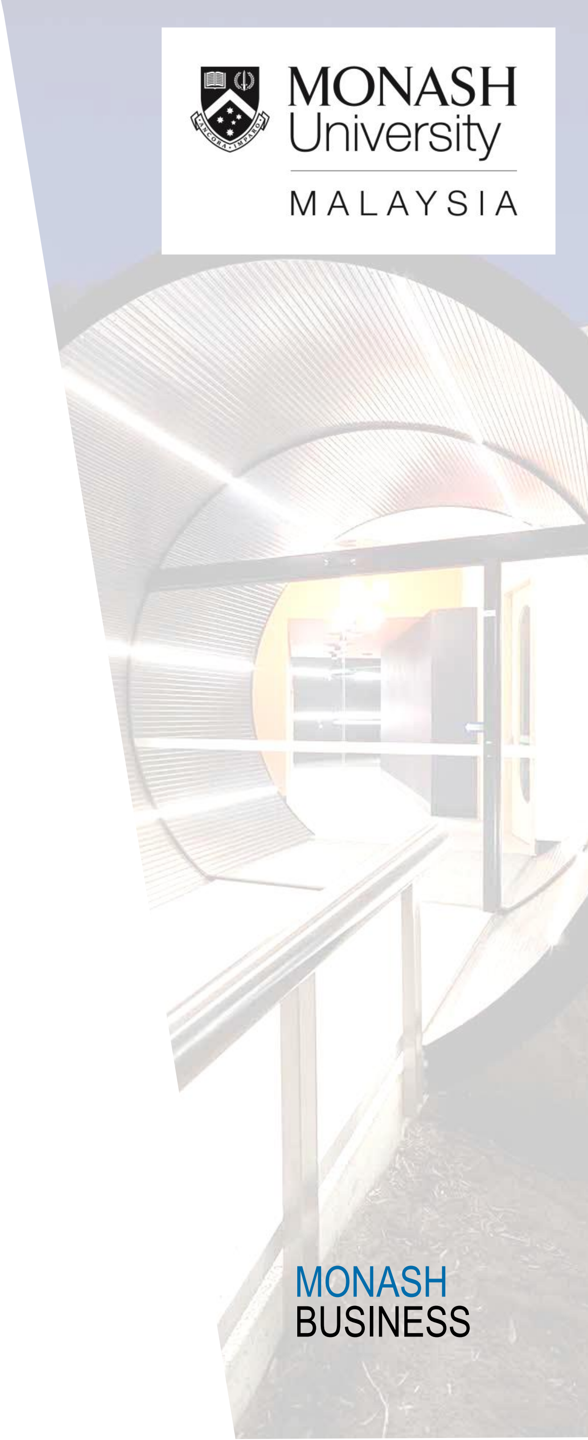
Unite

`unite()` can be used to rejoin the century and year columns that we created in the last example. It takes a data frame, the name of the new variable to create, and a set of columns to combine, again specified in `dplyr::select()` style.

```
table5 %>%
  unite(new, century, year)
```

A tibble: 6 × 3

country	new	rate
<chr>	<chr>	<chr>
Afghanistan	19_99	745/19987071
Afghanistan	20_00	2666/20595360
Brazil	19_99	37737/172006362
Brazil	20_00	80488/174504898
China	19_99	212258/1272915272
China	20_00	213766/1280428583



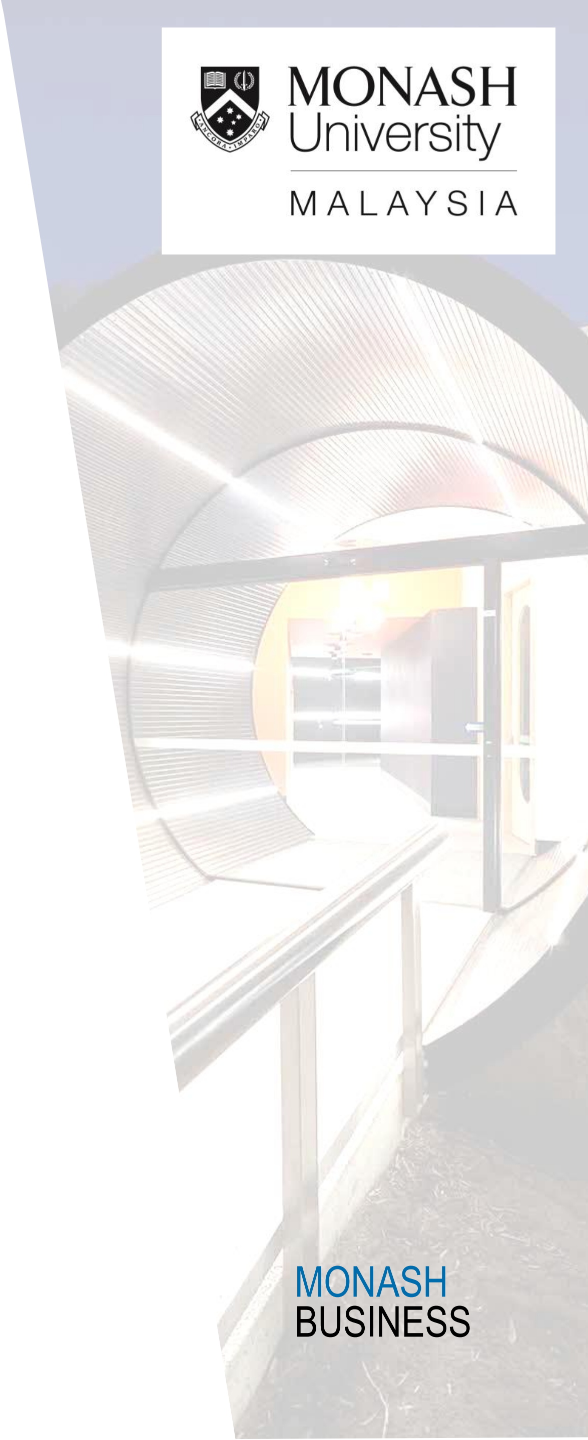
Unite

In this case, we also need to use the sep argument. The default will place an underscore (_) between the values from different columns. As we don't want any separator, use "“.

```
table5 %>%
  unite(new, century, year, sep = "")
```

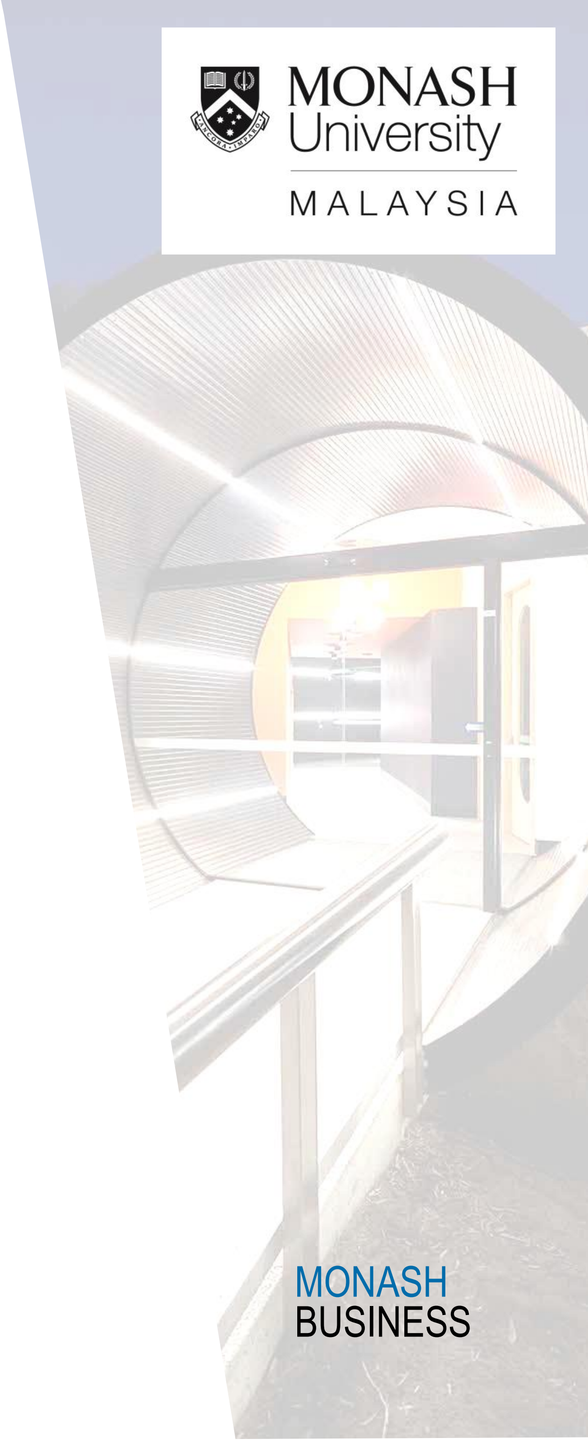
A tibble: 6 × 3

country	new	rate
<chr>	<chr>	<chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

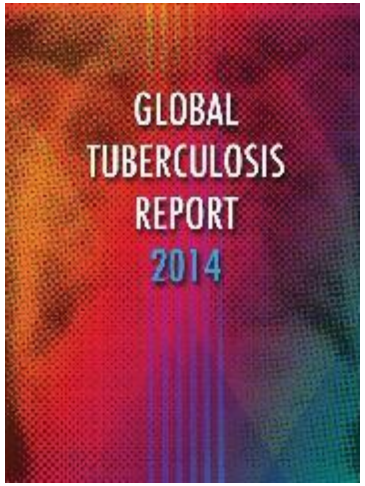


Outline

- ✓ Tidy Data with tidyr
    - ✓ Tidy Data
    - ✓ Pivoting
    - ✓ Separating and Uniting
    - ❑ **Case Study**



WHO



For the case study, the data from the 2014 World Health Organization (WHO) Global Tuberculosis (TB) Report is used.

The `tidyr::who` dataset contains TB cases broken down by year, country, age, gender, and diagnosis method.

While there is wealth of epidemiological information in this dataset, it's **challenging** to work with the data in the form that it's provided.

```
who
```

A tibble: 7240 × 60

country	iso2	iso3	year	new_sp_m014	new_sp_m1524	new_sp_m2534	new_sp_m3544	new_sp_m4554	new_sp_m5564	⋯	newrel_m4554	newrel_m5564
<chr>	<chr>	<chr>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	⋯	<int>	<int>
Afghanistan	AF	AFG	1980	NA	NA	NA	NA	NA	NA	⋯	NA	NA
Afghanistan	AF	AFG	1981	NA	NA	NA	NA	NA	NA	⋯	NA	NA
Afghanistan	AF	AFG	1982	NA	NA	NA	NA	NA	NA	⋯	NA	NA

MONASH University MALAYSIA

MONASH BUSINESS

WHO

The TB dataset from WHO is a typical real-life example, where it has redundant columns, odd variable codes, and many missing values. In short, who is messy, and we'll need multiple steps to tidy it. tidyr is designed so that each function does one thing well.

To start, let's look at columns that are **not variables**.

- country, iso2, and iso3 are 3 variables that redundantly specify the country,
- year is clearly also a variable.

We don't know what all the other columns are yet, but given the structure in the variable names (e.g. new_sp_m014, new_ep_m014, new_ep_f014) these are likely to be values, not variables.

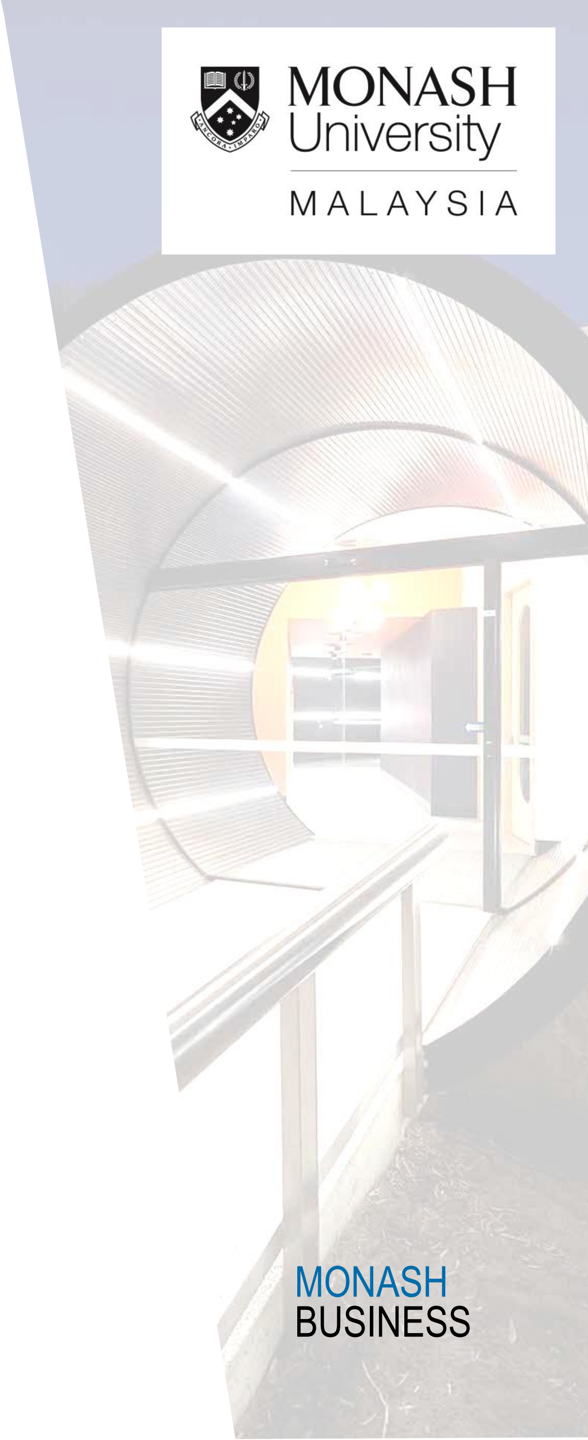
Longer

Let's gather all the columns from new_sp_m014 to newrel_f65. As we don't know what those values represent yet, we give them the generic name "key". We do know the cells represent the count of cases, so let's use the variable cases. In treating missing values, we will be using `values_drop_na` so we can focus on the values that are present.

```
who1 <- who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  )
who1
```

A tibble: 76046 × 6

country	iso2	iso3	year	key	cases
<chr>	<chr>	<chr>	<int>	<chr>	<int>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6



Count

Hints of the structure of values in the new key column can be found by counting.

```
who1 %>%
  count(key)
```

A tibble: 56 × 2

key	n
<chr>	<int>
new_ep_f014	1032
new_ep_f1524	1021
new_ep_f2534	1021
new_ep_f3544	1021
new_ep_f4554	1017

MONASH University MALAYSIA

MONASH BUSINESS

Parse

There is a data dictionary which you can use. It tells us that first 3 letters of each column denote whether the column contains new or old cases of TB. In this dataset, each column contains new cases.

The next two letters describe the type of TB:

rel: cases of relapse

ep: cases of extrapulmonary TB

sn: cases of pulmonary TB that could not be diagnosed by a pulmonary smear (smear negative)

sp: cases of pulmonary TB that could be diagnosed by a pulmonary smear (smear positive)

The 6th letter gives the sex of TB patients, males (m) and females (f).

Groups

The remaining numbers gives the age group, which groups into 7 age groups.

014: 0 – 14 years old
1524: 15 – 24 years old
2534: 25 – 34 years old
3544: 35 – 44 years old
4554: 45 – 54 years old
5564: 55 – 64 years old
65: 65 or older

MONASH University MALAYSIA

MONASH BUSINESS

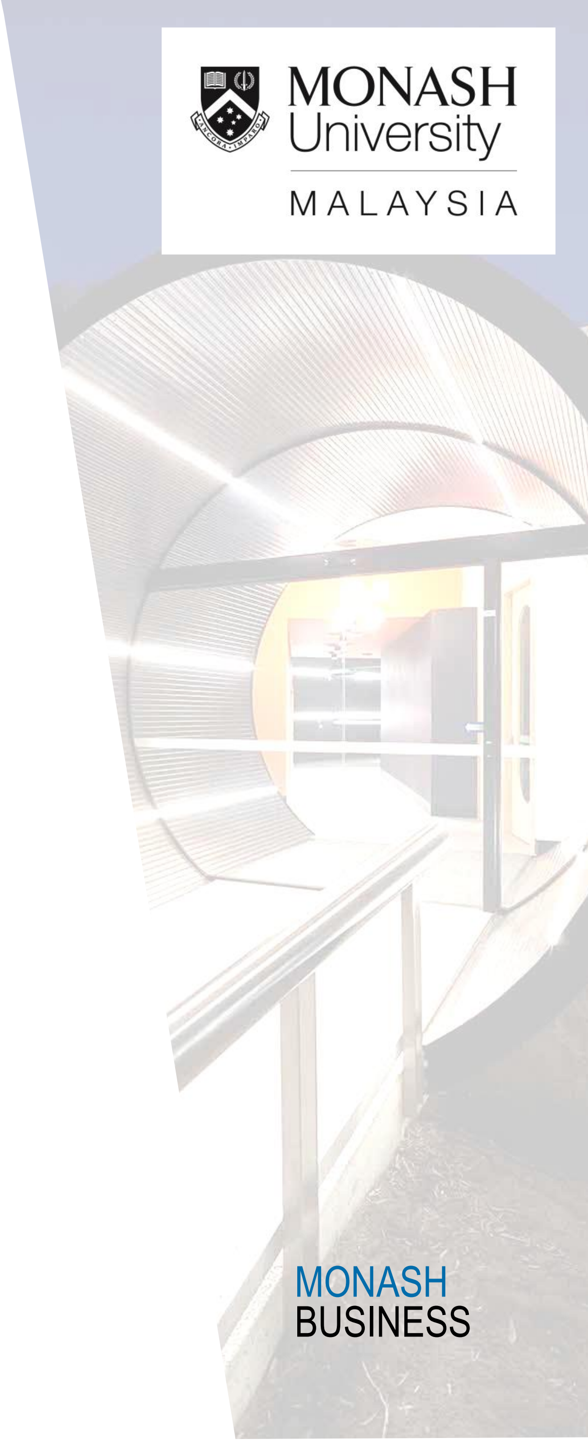
Mutate

A minor fix is needed to the format of the column names: unfortunately the names are slightly **inconsistent** because instead of new_rel we have newrel. `str_replace()` in strings, but the basic idea is pretty simple: replace the characters “newrel” with “new_rel”.   This makes all variable names consistent.

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
who2
```

A tibble: 76046 × 6

country	iso2	iso3	year	key	cases
<chr>	<chr>	<chr>	<int>	<chr>	<int>
Afghanistan	AF	AFG	1997	new_sp_m014	0
Afghanistan	AF	AFG	1997	new_sp_m1524	10
Afghanistan	AF	AFG	1997	new_sp_m2534	6



Separate

Values in each code can be separated with two passes of `separate()`. The first pass will split the codes at each underscore.

```
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
who3
```

A tibble: 76046 × 8

country	iso2	iso3	year	new	type	sexage	cases
<chr>	<chr>	<chr>	<int>	<chr>	<chr>	<chr>	<int>
Afghanistan	AF	AFG	1997	new	sp	m014	0
Afghanistan	AF	AFG	1997	new	sp	m1524	10
Afghanistan	AF	AFG	1997	new	sp	m2534	6

Count

We might as well **drop** the **new** column because it's constant in this dataset.

```
who3 %>%
  count(new)
who3
```

A tibble: 1 × 2

new	n
<chr>	<int>
new	76046

A tibble: 76046 × 8

country	iso2	iso3	year	new	type	sexage	cases
<chr>	<chr>	<chr>	<int>	<chr>	<chr>	<chr>	<int>
Afghanistan	AF	AFG	1997	new	sp	m014	0
Afghanistan	AF	AFG	1997	new	sp	m1524	10
Afghanistan	AF	AFG	1997	new	sp	m2534	6

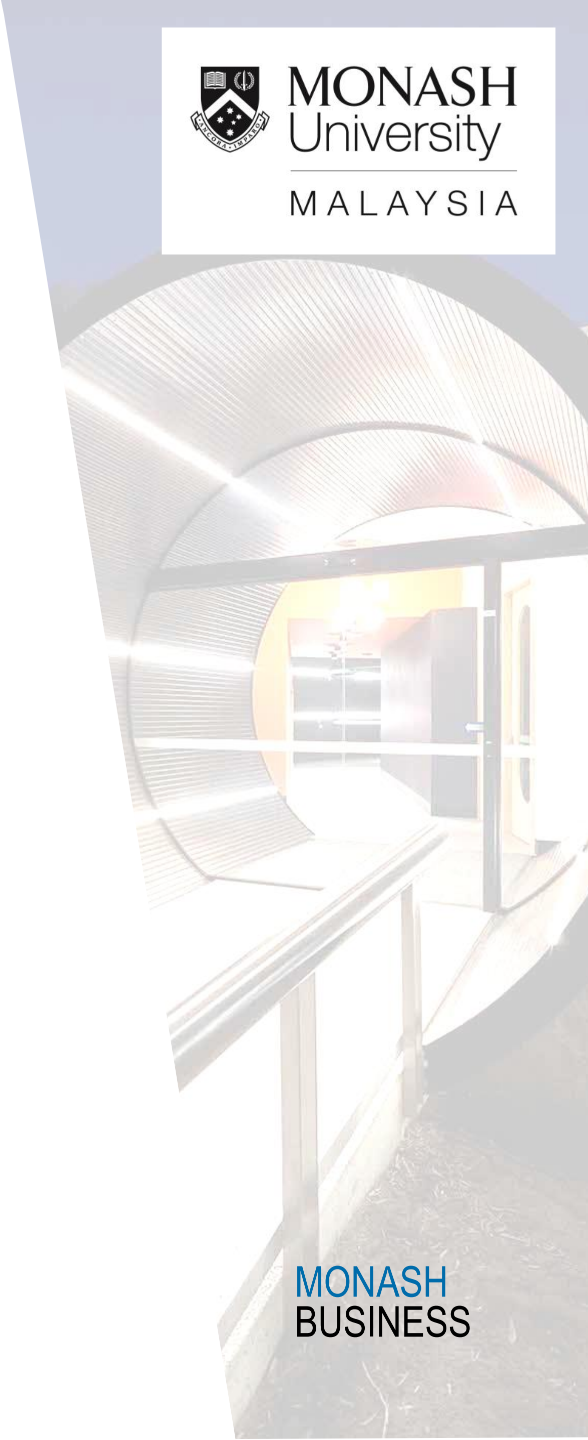
Count

While we're dropping columns, let's also **drop** iso2 and iso3 since they're redundant.

```
who4 <- who3 %>%
  select(-new, -iso2, -iso3)
who4
```

A tibble: 76046 × 5

country	year	type	sexage	cases
<chr>	<int>	<chr>	<chr>	<int>
Afghanistan	1997	sp	m014	0
Afghanistan	1997	sp	m1524	10
Afghanistan	1997	sp	m2534	6
Afghanistan	1997	sp	m3544	3
Afghanistan	1997	sp	m4554	5
Afghanistan	1997	sp	m5564	2



Separate

Next, separate sexage into sex and age by splitting after the first character.

```
who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
who5
```

A tibble: 76046 × 6

country	year	type	sex	age	cases
<chr>	<int>	<chr>	<chr>	<chr>	<int>
Afghanistan	1997	sp	m	014	0
Afghanistan	1997	sp	m	1524	10
Afghanistan	1997	sp	m	2534	6
Afghanistan	1997	sp	m	3544	3
Afghanistan	1997	sp	m	4554	5
Afghanistan	1997	sp	m	5564	2

MONASH University MALAYSIA

MONASH BUSINESS

Done!

The **who** dataset is now tidy! A complex pipe has been gradually built up.

```
who %>%
  pivot_longer(
    cols = new_sp_m014:newrel_f65,
    names_to = "key",
    values_to = "cases",
    values_drop_na = TRUE
  ) %>%
  mutate(
    key = stringr::str_replace(key, "newrel", "new_rel")
  ) %>%
  separate(key, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

A tibble: 76046 × 6

country	year	var	sex	age	cases
<chr>	<int>	<chr>	<chr>	<chr>	<int>
Afghanistan	1997	sp	m	014	0
Afghanistan	1997	sp	m	1524	10
Afghanistan	1997	sp	m	2534	6

MONASH University MALAYSIA

MONASH BUSINESS

Questions

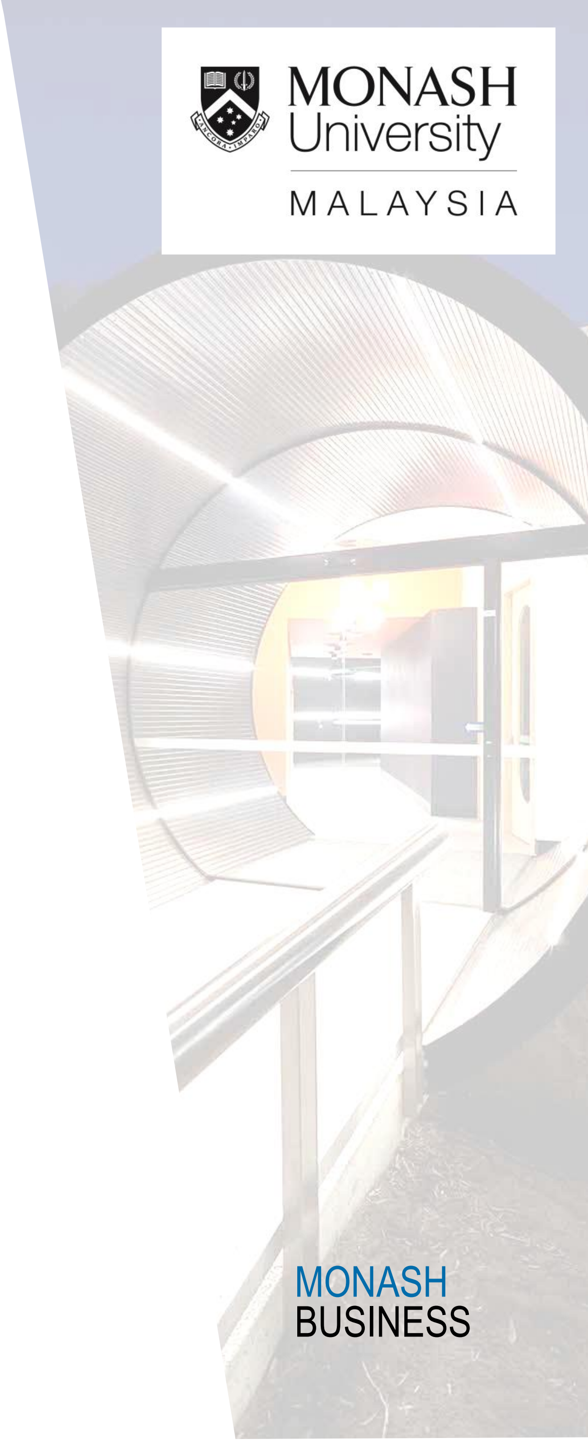
Confirm claim that **iso2** and **iso3** are redundant with **country**.

If iso2 and iso3 are redundant with country, then, within each country, there should only be one distinct combination of iso2 and iso3 values, which is the case.

```
select(who3, country, iso2, iso3) %>%
  distinct() %>%
  group_by(country) %>%
  filter(n() > 1)
```

A grouped_df: 0 × 3

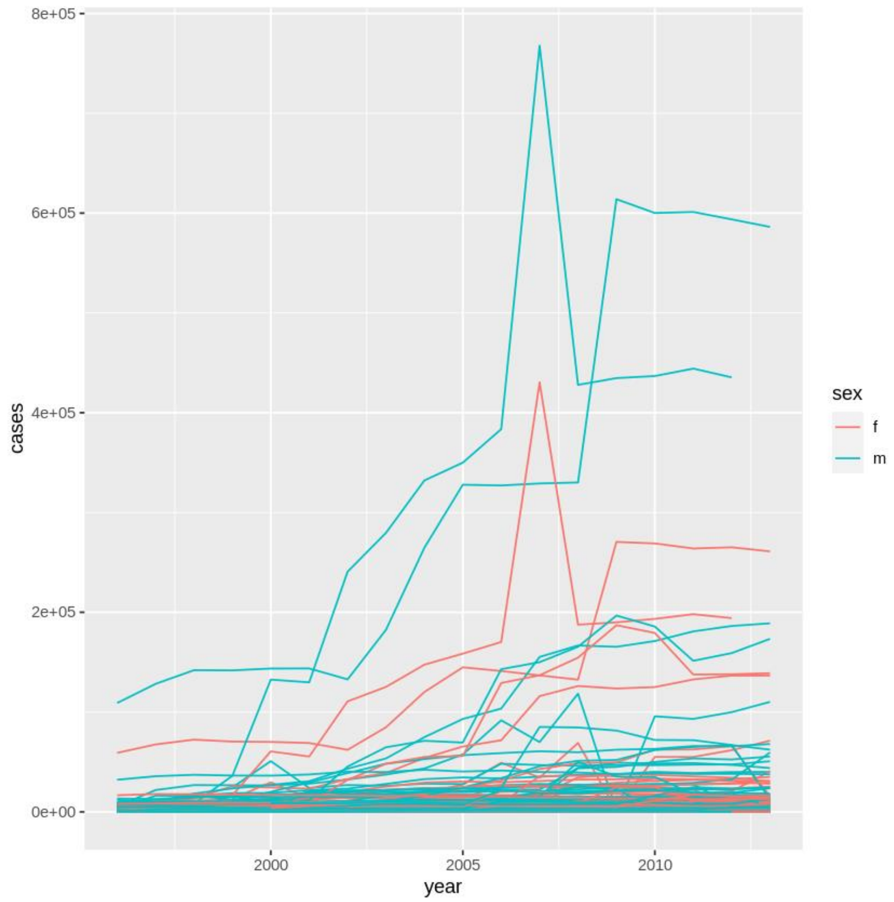
country	iso2	iso3
<chr>	<chr>	<chr>



Questions

For each country, year, and sex, compute the total number of cases of TB. Create an informative visualization of the data.

```
who5 %>%
  group_by(country, year, sex) %>%
  filter(year > 1995) %>%
  summarise(cases = sum(cases)) %>%
  unite(country_sex, country, sex, remove = FALSE) %>%
  ggplot(aes(x = year, y = cases, group = country_sex, colour = sex)) +
  geom_line()
```



MONASH BUSINESS

THANK YOU

FIND OUT MORE AT MONASH.EDU.MY
LIKE @MONASH UNIVERSITY MALAYSIA ON FACEBOOK
FOLLOW @MONASHMALAYSIA ON TWITTER