

INTRODUCTION TO DATA SCIENCE

Lecture 7

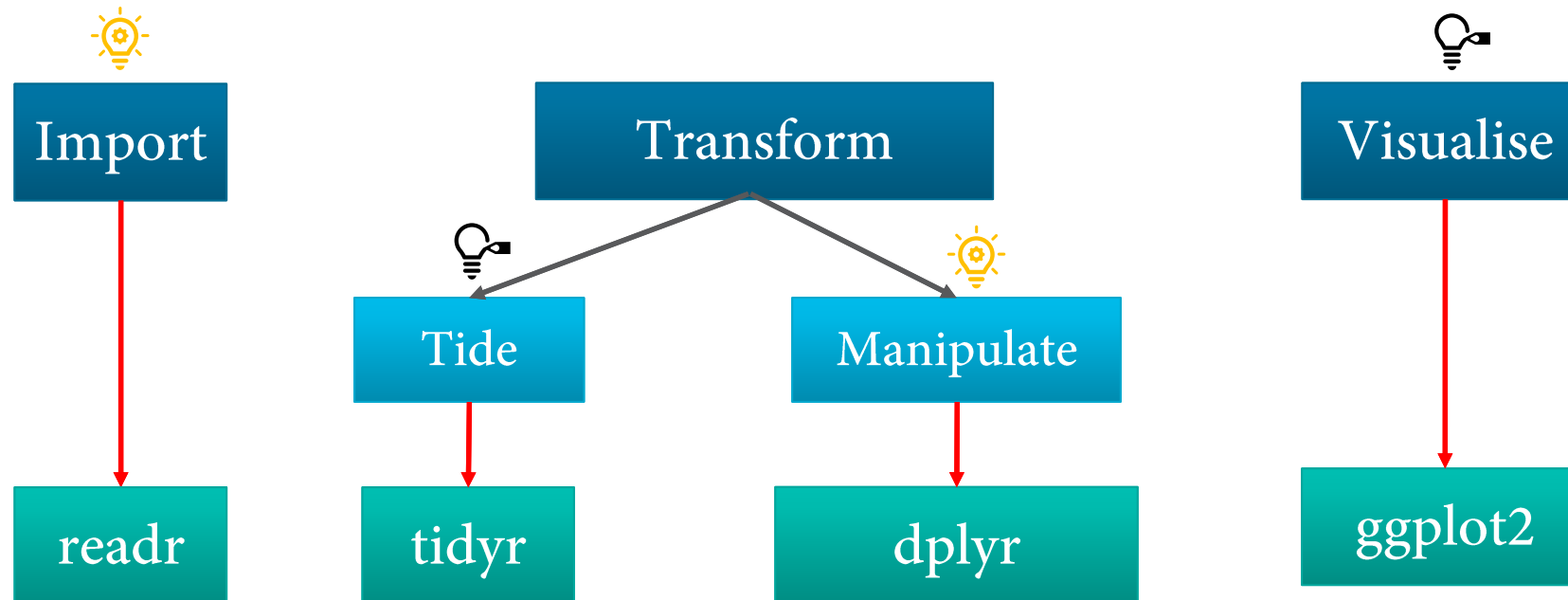
Dr. Ibrahim Radwan

OUTLINE

- Data Wrangling (Recap)
- Tidy data
- A case study

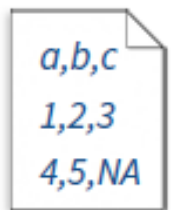
DATA WRANGLING

- Practically, we have three main processes to wrangle the data



DATA IMPORT (RECAP)

```
read_(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),  
      quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,  
      n_max), progress = interactive())
```

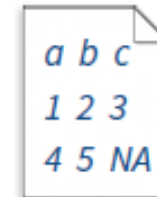


a,b,c
1,2,3
4,5,NA



A	B	C
1	2	3
4	5	NA

Comma Delimited Files
`read_csv("file.csv")`

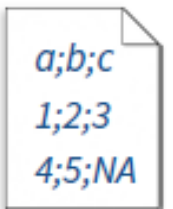


a b c
1 2 3
4 5 NA



A	B	C
1	2	3
4	5	NA

Tab Delimited Files
`read_tsv("file.tsv")`
Also `read_table()`

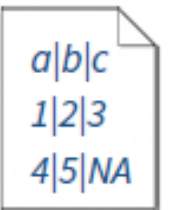


a;b;c
1;2;3
4;5;NA



A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
`read_csv2("file2.csv")`



a|b|c
1|2|3
4|5|NA



A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
`read_delim("file.txt", delim = "|")`

To save data into csv or txt file

Comma delimited file

`write_csv(x, path, na = "NA", append = FALSE,
col_names = !append)`

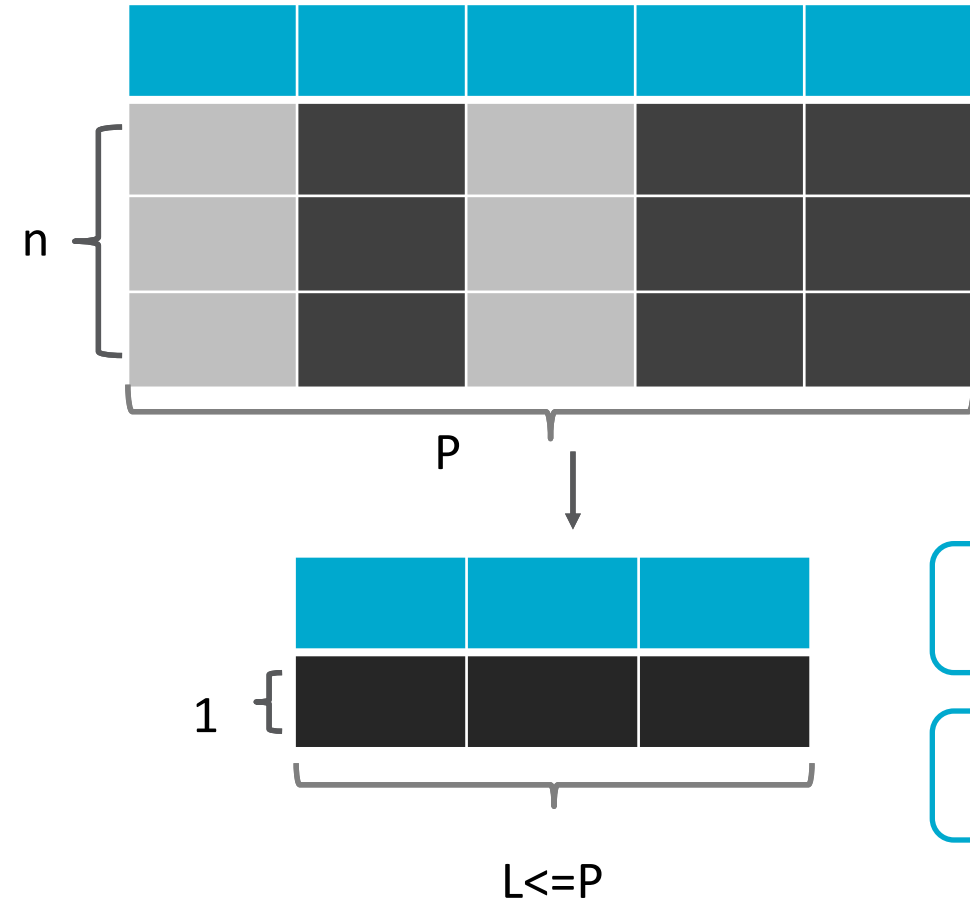
File with arbitrary delimiter

`write_delim(x, path, delim = " ", na = "NA",
append = FALSE, col_names = !append)`

DATA MANIPULATION (RECAP)

- The `dplyr` package in `tidyverse` library presents five verbs for manipulating the data in data frames:
 1. `filter()` extracts a subset of the rows (i.e., observations) based on some criteria
 2. `select()` extracts a subset of the columns (i.e., features, variables) based on some criteria
 3. `mutate()` adds or modifies existing columns
 4. `arrange()` sorts the rows
 5. `summarise()` aggregates the data across rows (e.g., group them according to some criteria)
- Each of these functions takes a data frame as its first argument and returns a data frame.

AGGREGATE (SUMMARISE)



```
summarise(dataframe, agg_func(col_name))
```

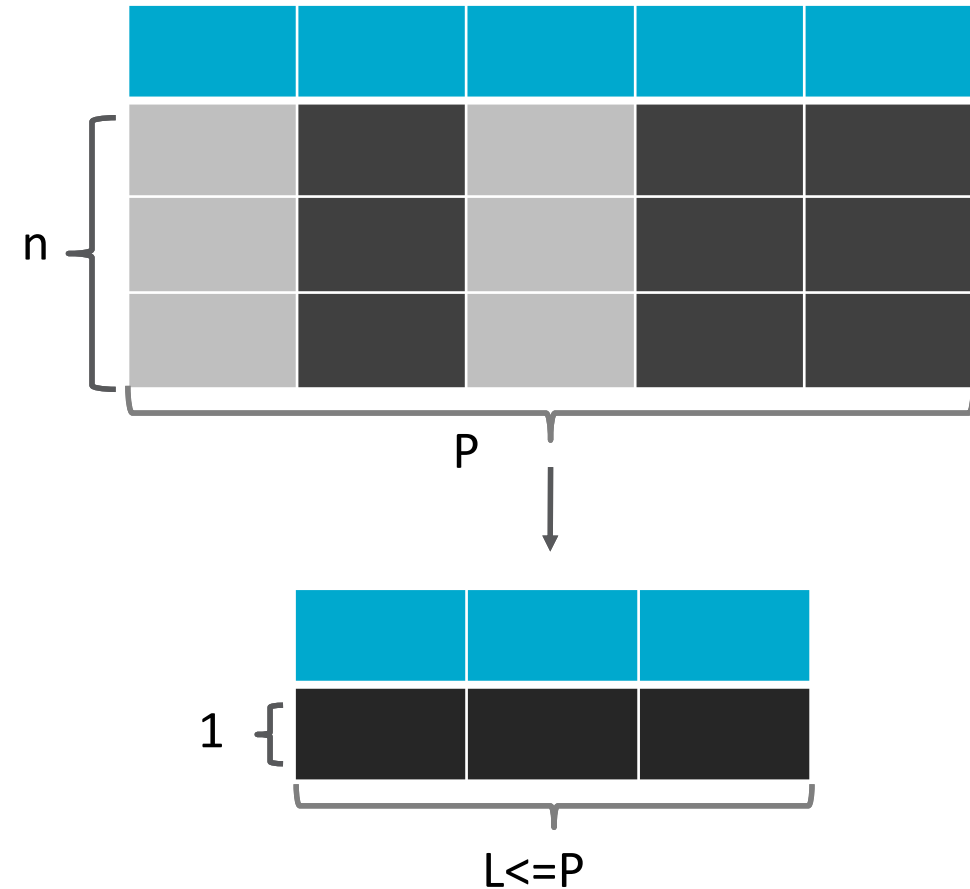
```
# call the required libraries
library(tidyverse)
library(nycflights13)
df <- flights
# extract a statistical metric from variable /
variables of the data
summarise(df, delay = mean(dep_delay, na.rm = TRUE))
```

Aggregation functions such as mean, sd, var, median, min and max

There is also summarise_each(dataframe, funs(aggregation_func))

summarise() is not terribly useful unless we pair it with
group_by()

AGGREGATE WITH GROUPING



```
group_by(dataframe, col_name)
```

```
# group the data of the flights by the date
by_day <- group_by(flights, year, month, day)
# get the average delay per date/day
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
# Imagine that we want to explore the relationship
between the distance and average delay for each
location.
```

```
by_dest <- group_by(flights, dest)
```

```
# extract the number of flights, average distance and
average delay for each destination
```

```
delay <- summarise(by_dest, count= n(), dist=
mean(distance, na.rm = TRUE), delay= mean(arr_delay,
na.rm = TRUE) )
```

```
# visualise to understand the relationship
```

```
ggplot(data= delay, mapping=aes(x= dist, y= delay)) +
geom_point(aes(size= count), alpha= 1/ 3) +
geom_smooth()
```

PIPE OPERATOR %>%

- In data wrangling, most likely, you need to perform *series* of operations (i.e. *verbs*) on the same data.
- This will need you to create intermediate tables temporarily to save the results to be processed with the next operations.
- R provides an elegant way to perform series of operations on the same data in one go via using the *pipe operator* %>%

original data → select → filter

```
F(x) is the same as  
x %>% F
```

```
16 %>% sqrt() %>% log2()  
[1] 2
```


THE PIPE OPERATOR %>% (2)

- In a previous ``nycflights`` example, there were three steps to extract the relationship between the distance and the delay of the flights per destination.
 1. Group flights by destination.
 2. Summarise to compute distance, average delay, and number of flights.
 3. Filter to remove noisy points

This can be achieved by using the pipe operator:

```
delay <- df %>%  
group_by(dest) %>%  
summarise(count= n(), dist= mean(distance, na.rm = TRUE),  
delay= mean(arr_delay, na.rm = TRUE)) %>%  
filter(count > 20, dest != 'HNL')
```

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

Having our data in a tidy format is a crucial step for data manipulation and exploring

country	year	cases	population
Afghanistan	1999	18145	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	17206362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	128042583

variables

country	year	cases	population
Afghanistan	1999	18145	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	17206362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	128042583

observations

country	year	cases	population
Afghanistan	99	725	987071
Afghanistan	00	666	595360
Brazil	99	737	206362
Brazil	00	488	504898
China	99	2258	272915272
China	00	6766	42583

values

TIDY DATA (2)

- Example of non-tidy data:

	country	1960	1961	1962	1963	1964	1965
1	Germany	2.41	2.44	2.47	2.49	2.49	2.48
2	South Korea	6.16	5.99	5.79	5.57	5.36	5.16

The data are not tidy, why?

1. Each row includes several observations and
2. One of the variables, year, is stored in the header.

TIDY DATA (3)

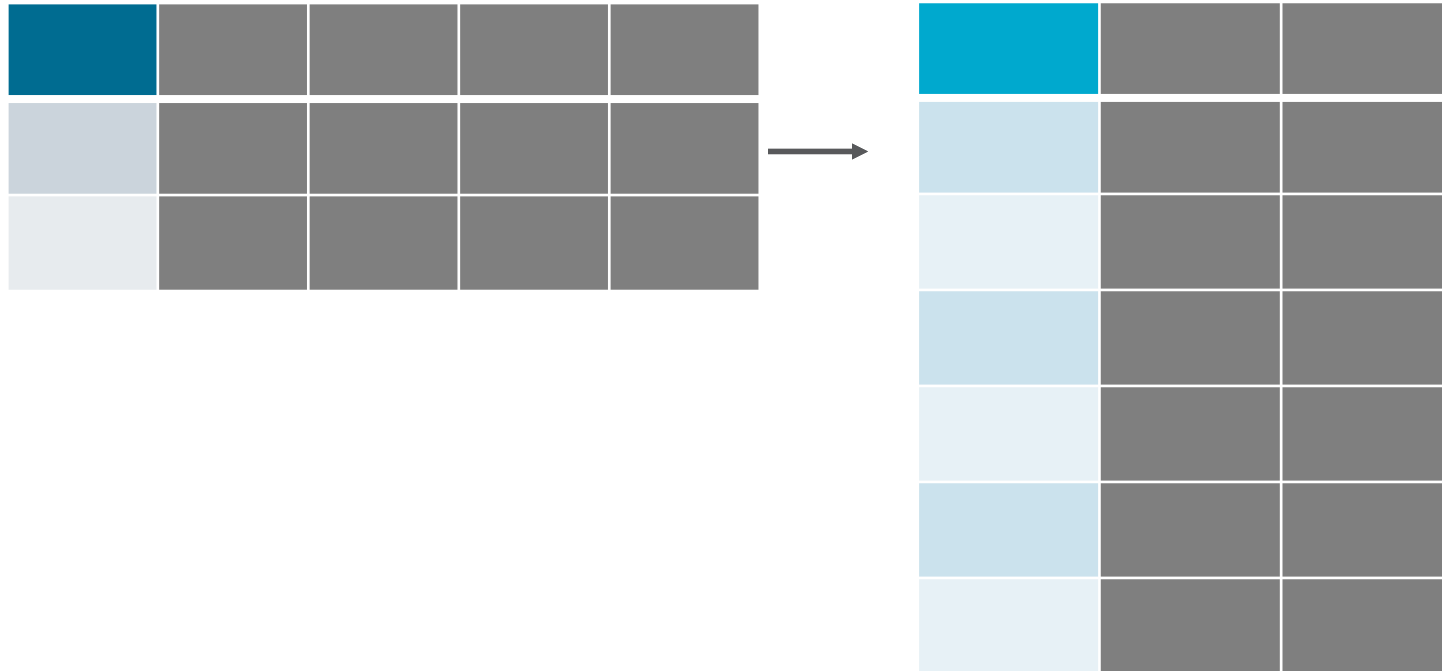
To make the data in previous slide tidy, we need to convert it from wide to long. To do so, we first define the variables embedded in the data. Here we have 3 variables. Then we tabulate the data within their corresponding variables.

Now, this dataset is tidy because each row presents one observation with the three variables being country, year and fertility rate.

index	country	year	fertility
1	Germany	1960	2.41
2	South Korea	1960	6.16
3	Germany	1961	2.44
4	South Korea	1961	5.99
5	Germany	1962	2.47
6	South Korea	1962	5.79
7	Germany	1963	2.49
8	South Korea	1963	5.57
9	Germany	1964	2.49
10	South Korea	1964	5.36
11	Germany	1965	2.48

- The ``tidyr`` package presents four main *verbs/functions* to tidy up the data:
 1. `gather()` collapses multiple columns into key-value pairs. It produces a “long” data format from a “wide” one.
 2. `spread()` takes two columns (key & value), and spreads into multiple columns: it makes “long” data wider. This is the reverse of `gather`.
 3. `unite()` unites multiple columns into one
 4. `separate()` takes a column and divides it into multiple columns
- Each of these functions takes a data frame as its first argument and returns a data frame.

GATHER DATA



```
gather(data, key, value, ...)
```

data: A data frame

key, value: Names of key and value columns to create in output

...: Specification of columns to gather. Allowed values are:
variable names

SPREAD DATA



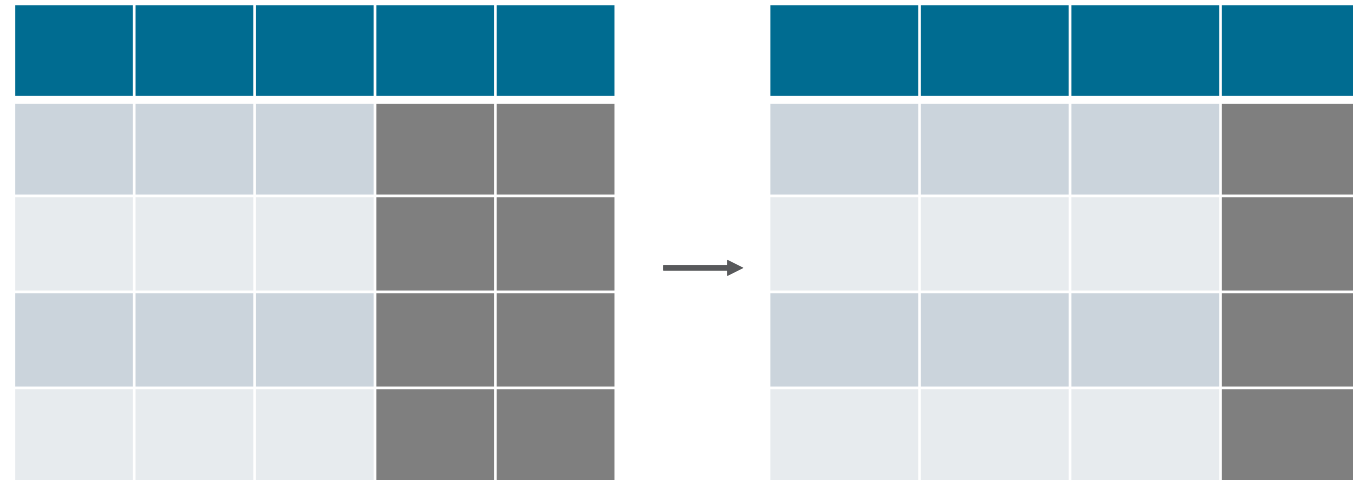

```
spread(data, key, value)
```

data: A data frame

key: The (unquoted) name of the column whose values will be used as column headings.

value: The (unquoted) names of the column whose values will populate the cells.

The function `unite()` takes multiple columns and paste them together into one-character column.



```
unite(data, col, ... , sep=)
```

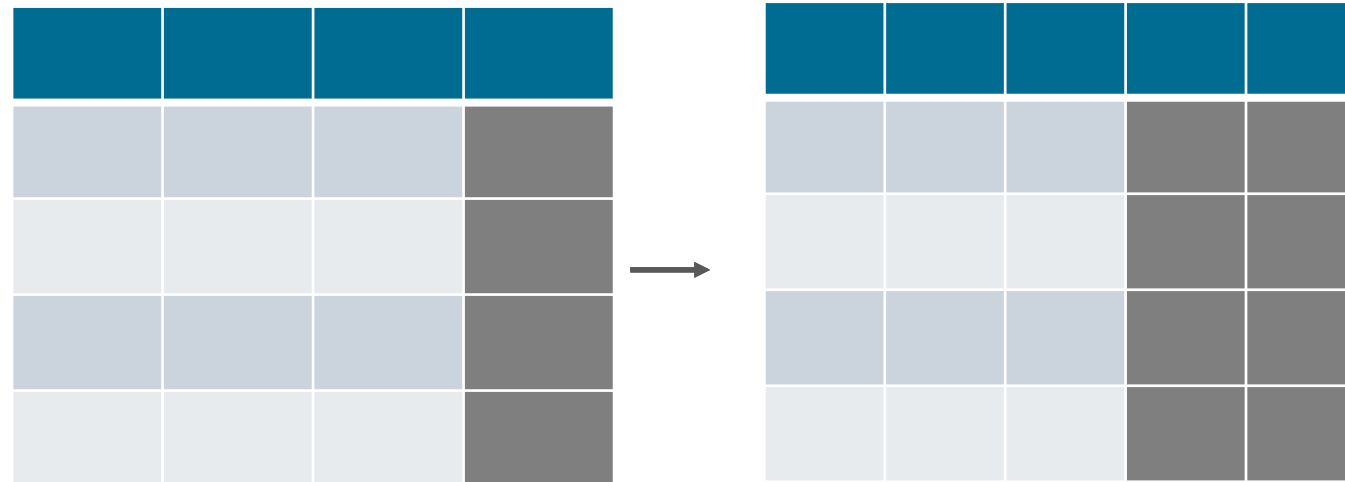
data: A data frame

col: The new (unquoted) name of column to add.

sep: Separator to use between values

SEPARATE DATA

The `separate()` function takes values inside a single character column and separates them into multiple columns.



```
separate(data, col, into, sep=)
```

data: A data frame

col: Unquoted column names

into: Character vector specifying the names of new variables to be created.

sep: Separator character between columns.

TIDY DATA (WRAP-UP)

- You should tidy your data for easier data analysis. The package *tidyr* provides the following functions.
 - Collapse multiple columns together into key-value pairs (long data format):
 - **gather**(data, key, value, ...)
 - Spread key-value pairs into multiple columns (wide data format):
 - **spread**(data, key, value)
 - Unite multiple columns into one:
 - **unite**(data, col, ...)
 - Separate one columns into multiple:
 - **separate**(data, col, into)

A CASE STUDY

- We have two datasets, which are downloaded from “*data.gov.au*” about the unemployment rate per gender for persons with disabilities between 1978 and 2017. They are separated into two files and we want to inspect the unemployment rate over different age groups.

We will do this task step-by-step in R-studio and the code can be downloaded from Canvas

RECOMMENDED READING

- You are recommended to read chapters 12 from the “*R for Data Science*” book:
 - <https://r4ds.had.co.nz/tidy-data.html>

ANNOUNCEMENTS

- Assignment 1 will be made available next week (Monday, week 8)
- Assignment 1 is due in week 10 (18th of April)
- Next week is the semester break, no classes.
- The week 9 online test is anticipated to be released Monday of week 9.
- ISEQ2 is open since yesterday for one week, let us hear from you all and better if you could leave a comment/feedback on how I and my team are doing in this unit?