Data Engineering
COMP2031/8031

- Topic Coordinator: Dr Mehwish Nasim
- Office: 3.13 Tonsley

Flinders
UNIVERSITY

Tidy data

- you will learn a consistent way to organise your data in R, an organisation called **tidy data**
- tidyr, a package that provides a bunch of tools to help tidy up your messy datasets. tidyr is a member of the core tidyverse

Tidy Data

```
table1
#> # A tibble: 6 x 4
#>   country      year  cases population
#>   <chr>       <int>  <int>      <int>
#> 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
table2
```

Flinders UNIVERSITY

```
table2
#> # A tibble: 12 x 4
#>    country      year type           count
#>    <chr>       <int> <chr>          <int>
#> 1 Afghanistan  1999 cases            745
#> 2 Afghanistan  1999 population  19987071
#> 3 Afghanistan  2000 cases           2666
#> 4 Afghanistan  2000 population  20595360
#> 5 Brazil       1999 cases           37737
#> 6 Brazil       1999 population 172006362
#> # … with 6 more rows
```

Flinders UNIVERSITY

```
table3
#> # A tibble: 6 x 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
```

```
# Spread across two tibbles
table4a  # cases
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#> * <chr>        <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil       37737  80488
#> 3 China       212258 213766
table4b  # population
#> # A tibble: 3 x 3
#>   country         `1999`     `2000`
#> * <chr>            <int>      <int>
#> 1 Afghanistan   19987071   20595360
#> 2 Brazil       172006362  174504898
#> 3 China       1272915272 1280428583
```

Flinders UNIVERSITY

Tibble

- These are all representations of the same underlying data, but they are not equally easy to use. One dataset, the tidy dataset, will be much easier to work with inside the tidyverse.
- There are three 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.

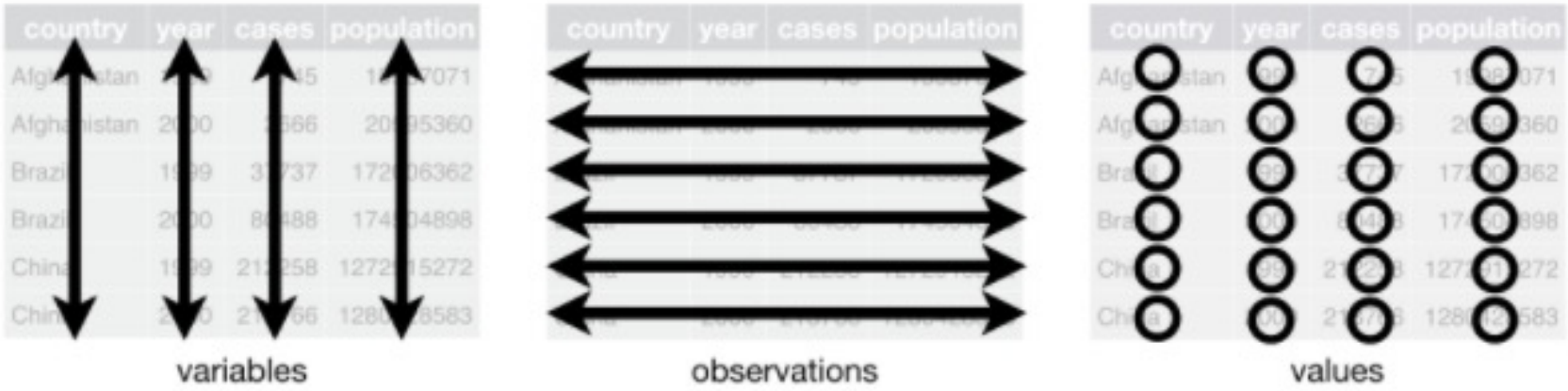

Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

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.

Why ensure that your data is tidy?

- There are two main advantages:
  - There's a 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.
  - There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine.

dplyr

Flinders UNIVERSITY

Data transformation

- library(nycflights13)
- library(tidyverse)

- To explore the basic data manipulation verbs of dplyr, we'll use nycflights13::flights. This data frame contains all 336,776 flights that departed from New York City in 2013. The data comes from the US Bureau of Transportation Statistics, and is documented in ?flights

flights

```
flights
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # … with 336,770 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
```

Copy

Flinders UNIVERSITY

Data frame

- You might notice that this data frame prints a little differently from other data frames you might have used in the past: it only shows the first few rows and all the columns that fit on one screen. (To see the whole dataset, you can run View(flights) which will open the dataset in the RStudio viewer). It prints differently because it's a **tibble**. Tibbles are data frames, but slightly tweaked to work better in the tidyverse.

abbreviations

- int stands for integers.
- dbl stands for doubles, or real numbers.
- chr stands for character vectors, or strings.
- dttm stands for date-times (a date + a time).

- lgl stands for logical, vectors that contain only TRUE or FALSE.
- fctr stands for factors, which R uses to represent categorical variables with fixed possible values.
- date stands for dates.

Dplyr basics

- learn the five key dplyr functions that allow you to solve the vast majority of your data manipulation challenges:
  - Pick observations by their values (filter()).
  - Reorder the rows (arrange()).
  - Pick variables by their names (select()).
  - Create new variables with functions of existing variables (mutate()).
  - Collapse many values down to a single summary (summarise()).
- These can all be used in conjunction with group_by() which changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

- These six functions provide the verbs for a language of data manipulation.
- All verbs work similarly:
- The first argument is a data frame.
- The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
- The result is a new data frame.
- Together these properties make it easy to chain together multiple simple steps to achieve a complex result. Let's dive in and see how these verbs work.

filter()

- filter() allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame. For example, we can select all flights on January 1st with:

```
filter(flights, month == 1, day == 1)
#> # A tibble: 842 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # … with 836 more rows, and 11 more variables: arr_delay <dbl>, carrier
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <db
#> #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Copy

- R either prints out the results, or saves them to a variable. If you want to do both, you can wrap the assignment in parentheses:

- (dec25 <- filter(flights, month == 12, day == 25))

- To use filtering effectively, you have to know how to select the observations that you want using the comparison operators. R provides the standard suite: >, >=, <, <=, != (not equal), and == (equal).
- When you're starting out with R, the easiest mistake to make is to use = instead of == when testing for equality. When this happens you'll get an informative error:

- Multiple arguments to filter() are combined with “and”: every expression must be true in order for a row to be included in the output. For other types of combinations, you’ll need to use Boolean operators yourself: & is “and”, | is “or”, and ! is “not”.

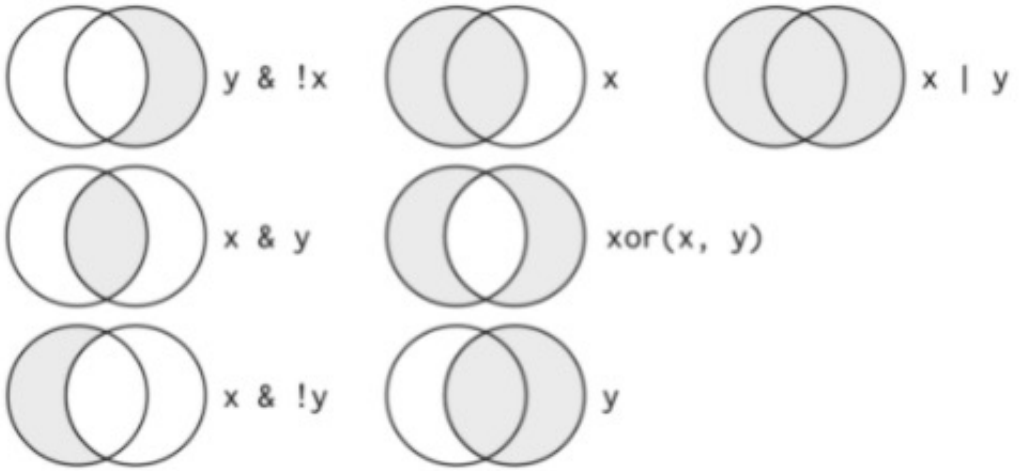

Figure 5.1: Complete set of boolean operations. `x` is the left-hand circle, `y` is the right-hand circle, and the shaded region show which parts each operator selects.

- The following code finds all flights that departed in November or December:
- filter(flights, month == 11 | month == 12)

- The order of operations doesn't work like English. You can't write filter(flights, month == (11 | 12)), which you might literally translate into "finds all flights that departed in November or December". Instead it finds all months that equal 11 | 12, an expression that evaluates to TRUE. In a numeric context (like here), TRUE becomes one, so this finds all flights in January, not November or December. This is quite confusing!
- A useful short-hand for this problem is x %in% y. This will select every row where x is one of the values in y. We could use it to rewrite the code above

- Sometimes you can simplify complicated subsetting by remembering De Morgan's law: !(x & y) is the same as !x | !y, and !(x | y) is the same as !x & !y. For example, if you wanted to find flights that weren't delayed (on arrival or departure) by more than two hours, you could use either of the following two filters:

- filter(flights, !(arr_delay > 120 | dep_delay > 120))
- filter(flights, arr_delay <= 120, dep_delay <= 120)

Arrange()

- arrange() works similarly to filter() except that instead of selecting rows, it changes their order.

- It takes a data frame and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns

- arrange(flights, year, month, day)

```
arrange(flights, year, month, day)
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_ar
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # … with 336,770 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
```

Copy

select()

- select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

- select(flights, year, month, day)