COMP824 2023 Week 7
Data Wrangling with `tidyr`

Department of Mathematical Sciences
Auckland University of Technology

Overview

The Process of Analytics

Data wrangling

Parsing

Writing data

Tidy Data

Example: Auckland Weather Data

Reading

Chapter 9, 12 Wickham and Grolemund (2020), R for Data Science https://r4ds.had.co.nz/

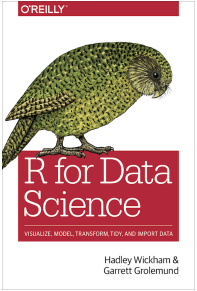
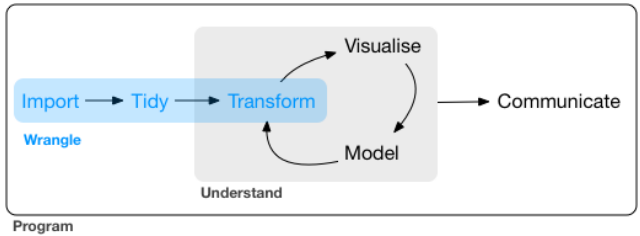


Figure 1: http://r4ds.had.co.nz/

The Process of Analytics



Learning objectives

- Understand what parsing is and how to specify variable types when importing data
- Know how to write data to a file
- Know the rules of tidy data and how to apply them in R using tidyverse functions

Data wrangling

Data wrangling

verb

The process of transforming and mapping data from one “raw” data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

Synonyms: data munging

Related terms: data cleaning, data transformation

Source: *Wikipedia*, https://en.wikipedia.org/wiki/Data_wrangling

Parsing

parsing $\approx$ interpreting

When R imports data is parses it to determine/assign a data type.

Parsing - Data Types

- Integers

```
str(parse_integer(c("1", "2", "3")))
```

```
 int [1:3] 1 2 3
```

- Dates

```
str(parse_date(c("2010-01-01", "1979-10-14")))
```

```
 Date[1:2], format: "2010-01-01" "1979-10-14"
```

Parsing - Errors

- Errors are produced if the data type does not match

```
str(parse_integer(c("1", "2", "3.5")))
```

```
 int [1:3] 1 2 NA
 - attr(*, "problems")= tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
  ..$ row     : int 3
  ..$ col     : int NA
  ..$ expected: chr "no trailing characters"
  ..$ actual  : chr "3.5"
```

```
str(parse_date(c("2010-01-01", "1979-10-:-)")))
```

```
 Date[1:2], format: "2010-01-01" NA
```

Parsing - Locale Specific

- Numbers

```
parse_number("$123.456.789")
```

```
[1] 123.456
```

```
parse_number("$123.456.789", locale=locale(grouping_mark = "."))
```

```
[1] 123456789
```

Parsing Files 1

- readr functions (e.g. `read_csv`) will "parse" the dataset to assign relevant data types for each column
- Can override default data types chosen by `read_csv`
- Customisation – e.g. specify column type

```
challenge <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_date()
  )
)
```

Parsing Files 2

```
tail(challenge, 5)
```

```
# A tibble: 5 x 2
      x y
  <dbl> <date>
1 0.164 2018-03-29
2 0.472 2014-08-04
3 0.718 2015-08-16
4 0.270 2020-02-04
5 0.608 2019-01-06
```

Overview

The Process of Analytics

Data wrangling

Parsing

Writing data

Tidy Data

Example: Auckland Weather Data

Writing data

Write a tibble `tb` to the file `myfile.csv`

```
write_csv(tb, "myfile.csv")
```

Other functions include:

- `write.table`
- `write.csv`
- `write.csv2`
- and others

Overview

The Process of Analytics

Data wrangling

Parsing

Writing data

Tidy Data

Example: Auckland Weather Data

tidyr package

Figure 3: https://https://www.tidyverse.org/

Three rules of tidy data

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

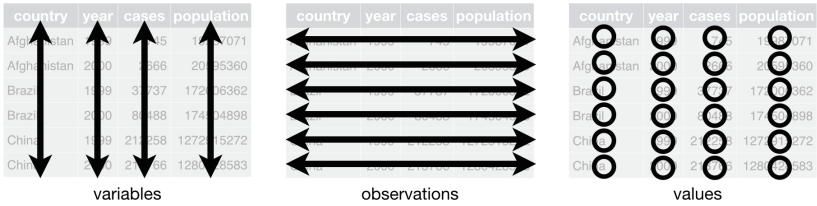


Figure 4: https://r4ds.had.co.nz/tidy-data.html

Tidy Data

Implementing tidy data in R

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

Why is this important?

- Consistency of data $\rightarrow$ fewer tools to learn
- R works well with vectors/columns of data

Example: Tidy data

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

```
table1
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
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
```

Quiz: Is this dataset tidy?

```
table2
```

```
# A tibble: 12 x 4
   country      year type           count
   <chr>       <dbl> <chr>          <dbl>
 1 Afghanistan  1999 cases            745
 2 Afghanistan  1999 population 19987071
 3 Afghanistan  2000 cases           2666
 # ... with 9 more rows
```

Quiz: Is this dataset tidy?

```
table3
```

```
# A tibble: 6 x 3
  country      year rate
  <chr>       <dbl> <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
```

Quiz: Are these datasets tidy?

```
table4a
```

```
# A tibble: 3 x 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

```
table4b
```

```
# A tibble: 3 x 3
  country          `1999`     `2000`
  <chr>             <dbl>      <dbl>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

Quiz: Which dataset/s are tidy? Answer

Table 1 is the only dataset that is “tidy”.

How to tidy data - key functions

- pivoting
- separating
- uniting
- missing values

Pivoting: Longer and/or Wider

- `pivot_longer`: one variable is spread across multiple columns
- `pivot_wider`: one observation is spread across multiple rows

Pivot Longer

```
table4a
```

```
# A tibble: 3 x 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Pivot Longer

```
table4a %>%
  pivot_longer(c(`1999`, `2000`), #selected column names
         names_to = "year",  #name of selected columns
         values_to = "cases" #name of the value in selected columns
         )
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <dbl>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
```

Pivot Wider

```
table2
```

```
# A tibble: 12 x 4
   country      year type           count
   <chr>       <dbl> <chr>          <dbl>
 1 Afghanistan  1999 cases            745
 2 Afghanistan  1999 population 19987071
 3 Afghanistan  2000 cases           2666
 # ... with 9 more rows
```

Pivot Wider

```
table2 %>%
    pivot_wider(names_from = type, #column of variable names
            values_from = count #column with values
            )
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
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
```

Separating and Uniting

- Separating: one column contains multiple variables
- Uniting: one variable is spread across multiple columns

Separating - original data

```
table3
```

```
# A tibble: 6 x 3
  country      year rate
  <chr>       <dbl> <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
```

Separating - tidy data

```
table3 %>%
  separate(rate, #column with multiple values
           into = c("cases", "population") )#names of new variables
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <dbl> <chr>  <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
```

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

Separating - new functions

`separate` has been superceded. It won't disappear but will only receive critical bug fixes. The new functions are:

- `seperate_wider_delim`
- `separate_wider_position`
- `separate_wider_regex`

Separating - separate_wider_delim

```
table3 %>%
  separate_wider_delim(
            cols = rate, #column with multiple values
            delim = "/",
            names = c("cases", "population") )#names of new variables
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <dbl> <chr>  <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
```

Uniting - original data

```
table5
```

```
# A tibble: 6 x 4
  country     century year  rate
  <chr>       <chr>   <chr> <chr>
1 Afghanistan 19      99    745/19987071
2 Afghanistan 20      00    2666/20595360
3 Brazil      19      99    37737/172006362
4 Brazil      20      00    80488/174504898
5 China       19      99    212258/1272915272
6 China       20      00    213766/1280428583
```

Uniting - tidier data

```
table5 %>%
  unite(new, #name of new column
        century, year #names of columns to unite
        )
```

```
# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <chr>
1 Afghanistan 19_99 745/19987071
2 Afghanistan 20_00 2666/20595360
3 Brazil      19_99 37737/172006362
4 Brazil      20_00 80488/174504898
5 China       19_99 212258/1272915272
6 China       20_00 213766/1280428583
```

Uniting - tidy data

```
table5 %>%
  unite(new, #name of new column
        century, year, #names of columns to unite
        sep = "" #separator
        )
```

```
# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <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
```

Missing Values

What values are missing from this dataset?

```
(stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)  ))
```

```
# A tibble: 7 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
# ... with 4 more rows
```

Missing Values - answer

Q4 2015 is missing, but so is Q1 2016.

Types of Missing Values

- Explicit – NA
- Implicit – not present

Missing Values: Explicit → Implicit

```
stocks %>%
  pivot_wider(names_from = year, values_from = return) %>%
  pivot_longer(cols = c(`2015`, `2016`),
               names_to = "year",
               values_to = "return",
               values_drop_na = TRUE )
```

```
# A tibble: 6 x 3
    qtr year  return
  <dbl> <chr>  <dbl>
1     1 2015    1.88
2     2 2015    0.59
3     2 2016    0.92
4     3 2015    0.35
5     3 2016    0.17
6     4 2016    2.66
```

Missing Values: Implicit → Explicit

```
stocks %>%
  complete(year, qtr)
```

```
# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
# ... with 5 more rows
```

Missing Values: Fill - original data

If previous values should be carried forward – `fill`

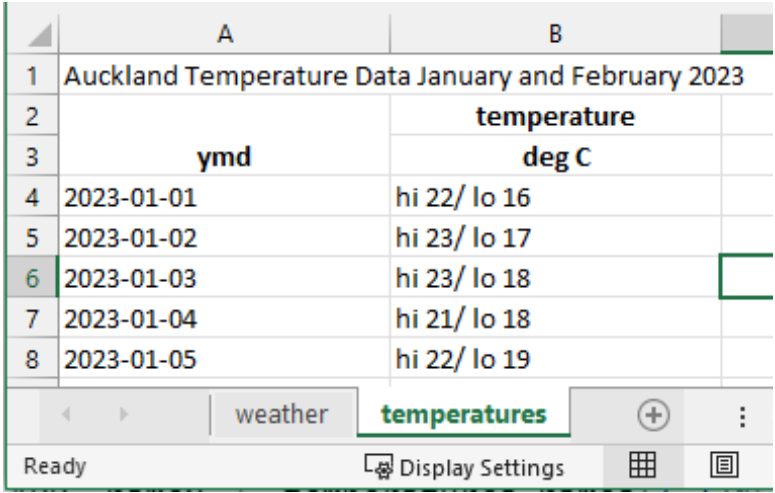
```
treatment <- tribble(
  ~ person,           ~ treatment, ~response,
  "Derrick Whitmore", 1,           7,
  NA,                 2,           10,
  NA,                 3,           9,
  "Katherine Burke",  1,           4
)
```

Missing Values: Fill - tidy data

```
treatment %>%
  fill(person)
```

```
# A tibble: 4 x 3
  person           treatment response
  <chr>                <dbl>    <dbl>
1 Derrick Whitmore         1        7
2 Derrick Whitmore         2       10
3 Derrick Whitmore         3        9
4 Katherine Burke          1        4
```

Example: Auckland Weather Data



	A	B
1	Auckland Temperature Data January and February 2023	
2		temperature
3	ymd	deg C
4	2023-01-01	hi 22/ lo 16
5	2023-01-02	hi 23/ lo 17
6	2023-01-03	hi 23/ lo 18
7	2023-01-04	hi 21/ lo 18
8	2023-01-05	hi 22/ lo 19

Figure 5: Screenshot of auckland_weather.xlsx

Import data

```
library(readxl)
fname <- "auckland_weather.xlsx"
temperatures <- read_excel(fname,
                           sheet = "temperatures",
                           skip = 2,
                           n_max = 59
                           #, range = A3:B62
                           )
```

```
# A tibble: 59 x 2
  ...1       `deg C`
  <chr>      <chr>
1 2023-01-01 hi 22/ lo 16
2 2023-01-02 hi 23/ lo 17
3 2023-01-03 hi 23/ lo 18
# ... with 56 more rows
```

Updating the column names - option 1

Rename manually

```
temperatures %>% rename(ymd = `...1`,
                        temperature = `deg C`)
```

```
# A tibble: 59 x 2
  ymd        temperature
  <chr>      <chr>
1 2023-01-01 hi 22/ lo 16
2 2023-01-02 hi 23/ lo 17
3 2023-01-03 hi 23/ lo 18
# ... with 56 more rows
```

Updating the column names - option 2

Rename using data from xlsx file

```
# Get names from file
(temperatures_names <- read_excel(fname,
                                  sheet = "temperatures",
                                  skip = 1, n_max = 2,
                                  col_names = FALSE))
```

```
# A tibble: 2 x 2
  ...1  ...2
  <chr> <chr>
1 ymd   temperature
2 <NA>  deg C
```

Updating the column names - option 2 (continued)

```
# Remove NAs
(temperatures_names <- temperatures_names %>%
              replace_na(replace = list(`...1` = '')) )
```

```
# A tibble: 2 x 2
  ...1   ...2
  <chr> <chr>
1 "ymd" temperature
2 ""    deg C
```

Updating the column names - option 2 (continued)

```
# Concatenate rows
library(stringr)
(col_names <- paste(temperatures_names[1,],
                    temperatures_names[2,]) %>%
              str_trim() %>%  #remove white space
              #replace spaces with _ (optional)
              str_replace_all(" ", "_") )
```

```
[1] "ymd"                "temperature_deg_C"
```

Updating the column names - option 2 (continued)

```
# Rename columns
(temperatures <- temperatures %>%
                    rename(!!col_names[1] := `...1`,
                           !!col_names[2] := `deg C`))
```

```
# A tibble: 59 x 2
  ymd        temperature_deg_C
  <chr>      <chr>
1 2023-01-01 hi 22/ lo 16
2 2023-01-02 hi 23/ lo 17
3 2023-01-03 hi 23/ lo 18
# ... with 56 more rows
```

Tidy the `temperatures` data - option 1

```
temperatures %>%
  tidyr::separate(temperature_deg_C,
                  into = c("high", "low"),
                  sep  = "/",
                  convert = TRUE) %>%
         mutate(high = str_replace(high, 'hi ', ""),
                low =  str_replace(low, ' lo ', "")) %>%
         type_convert()
```

```
# A tibble: 59 x 3
  ymd         high   low
  <date>     <dbl> <dbl>
1 2023-01-01    22    16
2 2023-01-02    23    17
3 2023-01-03    23    18
# ... with 56 more rows
```

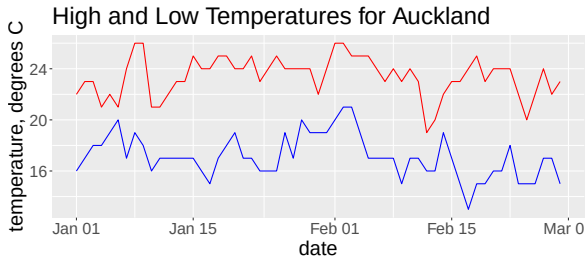
Tidy the `temperatures` data - option 2

```
(temperatures %>%
  tidyr::separate(temperature_deg_C,
                  into = c("temp", "high", "temp2", "low"),
                  sep = c("\\s+")  )  %>%
         mutate(high = str_replace(high, '/', "")) %>%
         select(-temp, -temp2) %>%
         type_convert() -> temperatures)
```

```
# A tibble: 59 x 3
  ymd         high   low
  <date>     <dbl> <dbl>
1 2023-01-01    22    16
2 2023-01-02    23    17
3 2023-01-03    23    18
# ... with 56 more rows
```

Plot high and low temperatures (from last week's lab)

```
temperatures %>%  ggplot() +
   geom_line(mapping = aes(x = ymd, y = high), col = "red") +
   geom_line(mapping = aes(x = ymd, y = low), col = "blue") +
  labs(x = "date",
       y = "temperature, degrees C",
       title = "High and Low Temperatures for Auckland")
```



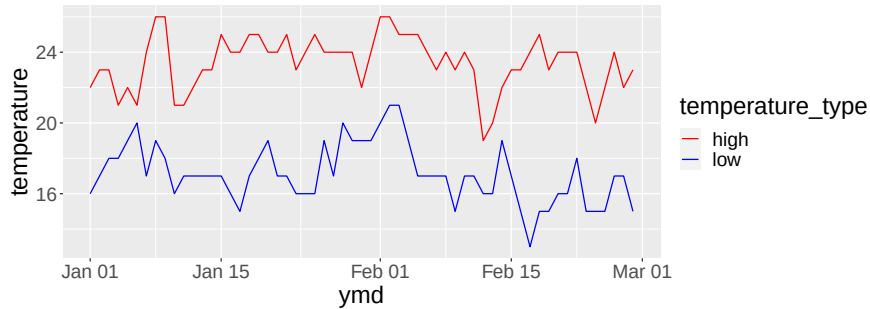
Plot high and low temperatures (a better way)

```
(temperatures_long <-
   temperatures %>% pivot_longer(c("high", "low"),
                                 names_to = "temperature_type",
                                 values_to = "temperature"))
```

```
# A tibble: 118 x 3
  ymd        temperature_type temperature
  <date>     <chr>                   <dbl>
1 2023-01-01 high                       22
2 2023-01-01 low                        16
3 2023-01-02 high                       23
# ... with 115 more rows
```

Plot high and low temperatures (a better way)

```
temperatures_long %>%  ggplot() +
                    geom_line(mapping = aes(x = ymd,
                                            y = temperature,
                                            col = temperature_type)) +
    # manual change colours (optional)
    scale_color_manual(breaks = c("high", "low"),
                       values=c("red", "blue"))
```



Summary

The Process of Analytics

Data wrangling

Parsing

Writing data

Tidy Data

Example: Auckland Weather Data

Learning objectives

- Understand what parsing is and how to specify variable types when importing data
- Know how to write data to a file
- Know the rules of tidy data and how to apply them in R using tidyverse functions

References

Wickham, Hadley, and Garrett Grolemund. 2020. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*.