Tabular Data

Data Aquisition and Distribution

Dr Zak Varty

1. Loading Tabular Data

Tabular Data Formats

Simpler, open source formats improve accessibility and reproducibility.

- `tabular-data.csv`
- `tabular-data.tsv`
- `tabular-data.txt`

.xlsx

Is not the only extension

Reading Tabular Data

`random-data.csv` contains 26 observations of 4 variables:

- `id`, a roman letter identifier;
- `gaussian`, standard normal random variates;
- `gamma`, gamma(1,1) random variates;
- `uniform`, uniform(0,1) random variates.

Equivalent data also stored in `random-data.tsv` and `random-data.txt`, using tab and space separation.

Reading Tabular Data - Base R

```r
random_df <- read.csv(file = 'random-data.csv')
print(random_df)
```

```
   id    gaussian      gamma    uniform
1   a -1.20706575 0.98899970 0.22484576
2   b  0.27742924 0.03813386 0.08498474
3   c  1.08444118 1.09462335 0.63729826
4   d -2.34569770 1.49301101 0.43101637
5   e  0.42912469 5.40361248 0.07271609
6   f  0.50605589 1.72386539 0.80240202
7   g -0.57473996 1.95357133 0.32527830
8   h -0.54663186 0.07807803 0.75728904
9   i -0.56445200 0.21198194 0.58427152
10  j -0.89003783 0.20803673 0.70883941
11  k -0.47719270 2.08607862 0.42697577
12  l -0.99838644 0.49463708 0.34357270
13  m -0.77625389 0.77171305 0.75911999
14  n  0.06445882 0.37216648 0.42403021
15  o  0.95949406 1.88207991 0.56088725
```

Output is a `data.frame` object. (List of vectors with some nice methods)

Reading Tabular Data - {readr}

```r
random_tbl <- readr::read_csv(file = 'random-data.csv')
print(random_tbl)
```

```
# A tibble: 26 × 4
   id    gaussian  gamma uniform
   <chr>    <dbl>  <dbl>   <dbl>
 1 a       -1.21  0.989   0.225
 2 b        0.277 0.0381  0.0850
 3 c        1.08  1.09    0.637
 4 d       -2.35  1.49    0.431
 5 e        0.429 5.40    0.0727
 6 f        0.506 1.72    0.802
 7 g       -0.575 1.95    0.325
 8 h       -0.547 0.0781  0.757
 9 i       -0.564 0.212   0.584
10 j       -0.890 0.208   0.709
# ℹ 16 more rows
```

Output is a `tibble` object. (List of vectors with some nicer methods)

Benefits of `readr::read_csv()`

1. Increased speed (approx. 10x) and progress bar.

2. Strings are not coerced to factors. No more `stringsAsFactors = FALSE`

3. No row names and nice column names.

4. Reproducibility bonus: does not depend on operating system.

WTF: Tibbles (Printing)

- Default to first 10 rows and as many columns as will comfortably fit on your screen.
- Can adjust this behaviour in the print call:

```r
# print first three rows and all columns
print(random_tbl, n = 3, width = Inf)
```

Bonus: Colour formatting in IDE and each column tells you it’s type.

WTF: Tibbles (Subsetting)

Subsetting tibbles will always return another tibble.

```
# Row Subsetting
random_tbl[1, ] # returns tibble
random_df[1, ]  # returns data.frame

# Column Subsetting
random_tbl[ , 1]      # returns tibble
random_df[ , 1]       # returns vector

# Combined Subsetting
random_tbl[1, 1]      # returns 1x1 tibble
random_df[1, 1]       # returns single value
```

Avoids edge cases associated with working on data frames.

Other {readr} functions

See {readr} documentation, additional arguments for reading messy data.

Reading Tabular Data:

```
library(readr)
read_tsv("random-data.tsv")
read_delim("random-data.txt", delim = " ")
```

Writing Tabular Data:

```
write_csv(random_tbl, "random-data-2.csv")
write_tsv(random_tbl, "random-data-2.tsv")
write_delim(random_tbl, "random-data-2.tsv", delim = " ")
```

When You Really Need Speed

Some times you have to load lots of large data sets, in which case a 10x speedup might not be sufficient.

If each data set still fits inside RAM, then check out `data.table::fread()` which is optimised for speed.

(Alternatives exist for optimal memory usage and data too large for working memory, but not covered here.)

2. Tidy Data

Wide vs Tall Data: Example 1

Person	Age	Weight	Height
Bob	32	168	180
Alice	24	150	175
Steve	64	144	165

Person	Variable	Value
Bob	Age	32
Bob	Weight	168
Bob	Height	180
Alice	Age	24
Alice	Weight	150
Alice	Height	175
Steve	Age	64
Steve	Weight	144
Steve	Height	165

[Source: Wikipedia - Wide and narrow data]

Wide vs Tall Data: Example 2

Team	Points	Assists	Rebounds
A	88	12	22
B	91	17	28
C	99	24	30
D	94	28	31

[Source: Statology - Long vs wide data]

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

Wide vs Tall Data

Wide Data

- First column has unique entries
- Easier for humans to read and compute on
- Harder for machines to compute on

Tall Data

- First column has repeating entries
- Harder for humans to read and compute on
- Easier for machines to compute on

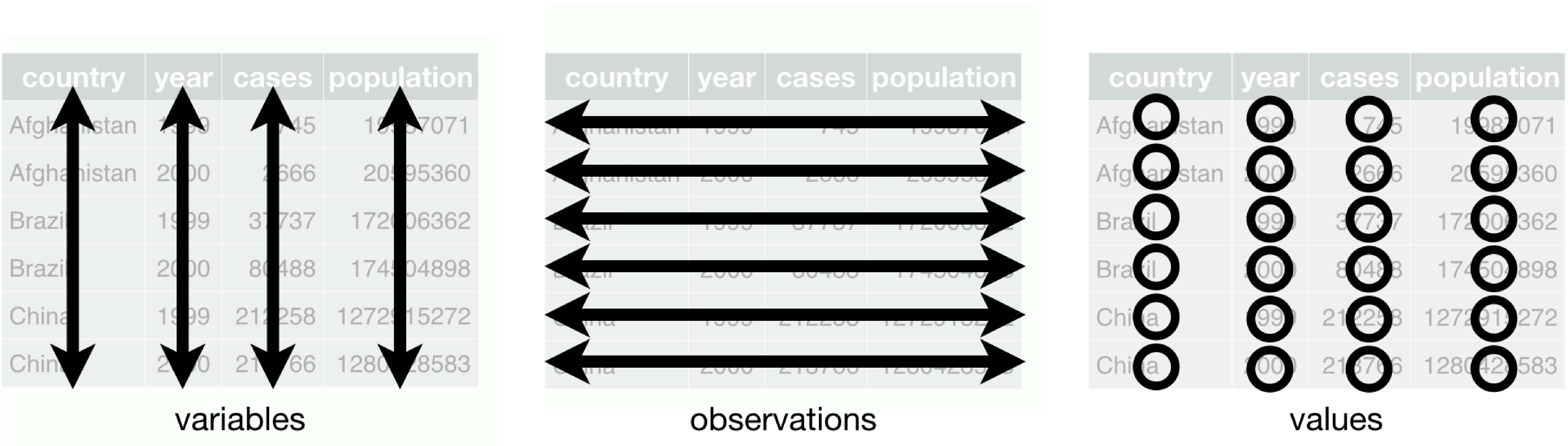
Pivoting Between Wide and Long Formats

- Error control at input and analysis is format-dependent.
- Switching between long and wide formats useful to control errors.
- Easy with the `{tidyr}` package functions

```r
tidyr::pivot_longer()
tidyr::pivot_wider()
```

(example coming up soon)

WTF: Tidy Data



[Image: R4DS - Chapter 12]

WTF: Benefits of Tidy Data

- **Consistent data format:** Reduces cognitive load and allows specialised tools (functions) to efficiently work with tabular data.

- **Vectorisation**: Keeping variables as columns allows for very efficient data manipulation.

(this goes back to data frames and tibbles being lists of vectors)

Example: Tidy Longer

Consider trying to plot these data as time series. The `year` variable is trapped in the column names!

```r
countries
```

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

To tidy this data, we need to `pivot_longer()`. We will turn the column names into a new `year` variable and retaining cell contents as a new variable called `cases`.

Example: Tidied Longer

```r
countries %>%
  tidyr::pivot_longer(cols = c(`1999`,`2000`), names_to = "year", values_to = "cases")
```

```
# A tibble: 6 × 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
```

Much better!

Example Tidy Wider

There are other times where we might have to widen our data to tidy it.

This example is not tidy. Why not?

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

Example Tidy Wider

There are other times where we might have to widen our data to tidy it.

This example is not tidy. Why not?

The observational unit here is a team.

Each variable should be a separate column, with cells containing their values.

Team	Variable	Value
A	Points	88
A	Assists	12
A	Rebounds	22
B	Points	91
B	Assists	17
B	Rebounds	28
C	Points	99
C	Assists	24
C	Rebounds	30
D	Points	94
D	Assists	28
D	Rebounds	31

Example: Tidied Wider

```r
tournament %>%
  tidyr::pivot_wider(
    id_cols = "Team",
    names_from = "Variable",
    values_from = "Value")
```

```
# A tibble: 4 × 4
  Team  Points Assists Rebounds
  <chr>  <dbl>   <dbl>    <dbl>
1 A         88      12       22
2 B         91      17       28
3 C         99      24       30
4 D         94      28       31
```

Other Helpful Functions

The `pivot_*()` functions resolve issues with rows (too many observations per row or rows per observation).

There are similar helper functions to solve column issues:

- Multiple variables per column: `tidyr::separate()`,
- Multiple columns per variable: `tidyr::unite()`.

Missing Values and Tidy Data

In tidy data, every cell contains a value. Including cells with missing values.

- Missing values are coded as `NA` (generic) or a type-specific `NA`, such as `NA_character_`.
- The `{readr}` family of `read_*()` function have good defaults and helpful `na` argument.
- Explicitly code `NA` values when collecting data, avoid ambiguity: " ", -999 or worst of all 0.
- More on missing values in EDA videos…

Wrapping Up

1. Reading in tabular data by a range of methods

2. Introduced the `tibble` and tidy data (+ tidy not always best)

3. Tools for tidying messy tabular data