Data Management and Manipulation

Lecture 02.3: Tidy Data

Lauren Sullivan

Module: Data Management, Visualization & Reproducibility

Readings

Required for class:

- NA

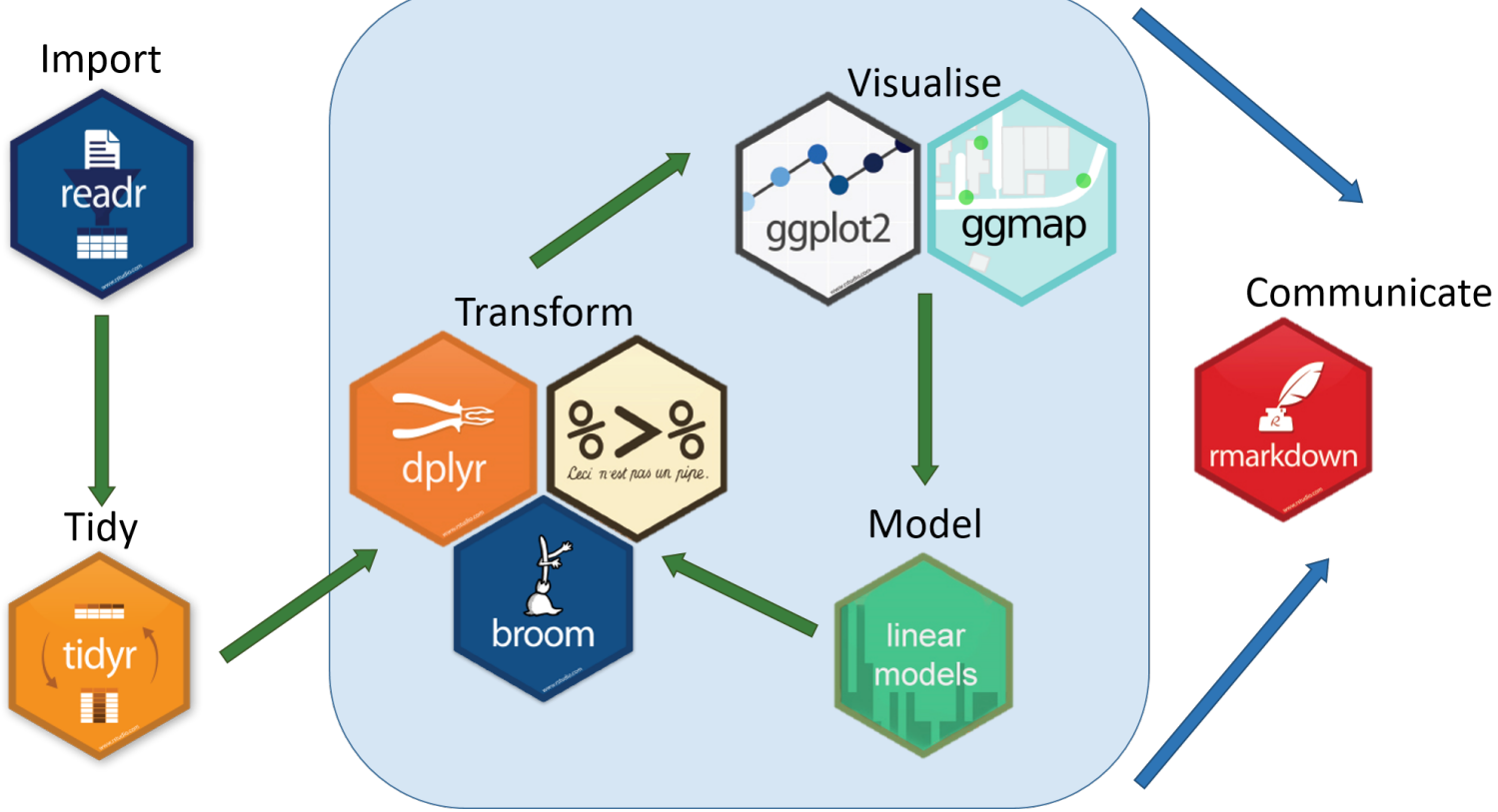
Optional:

- Tidyverse
- Grolemund & Wickham (2017) *R for Data Science. - Tidy Data with tidyr*
- Data Import::Cheat Sheet - for readr and tidyr

Data Manipulation Goals

- Perform all manipulation in R
- Preserves data integrity
- This will take a lot of time at first but is worth the effort
- Remember Google is your friend!

Tidy Data


Import
readr
Tidy
tidyr
Transform
dplyr
%>%
Leci n'est pas un pipe.
broom
Visualise
ggplot2
ggmap
Model
linear models
Communicate
rmarkdown

Tidy Data

- There are lots of ways to represent the same set of data in tables, but not all are tidy.

```
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
#> # 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
```

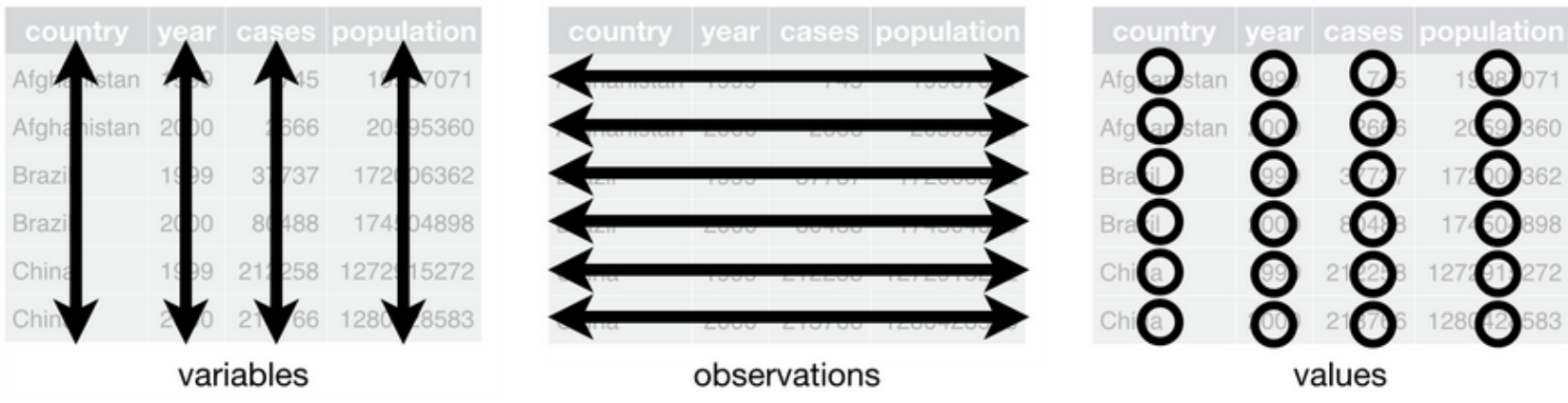
```
table3
#> # A tibble: 6 x 3
#>   country      year rate
#> * <chr>       <int> <chr>
#> 1 Afghanistan  1999 745/19987071
#> 2 Afghanistan  2000 2666/20595360
#> 3 Brazil       1999 37737/172006362
#> 4 Brazil       2000 80488/174504898
#> 5 China        1999 212258/1272915272
#> 6 China        2000 213766/1280428583
```

This is a dataset of tuberculosis cases in 1999 and 2000 from several countries from the World Health Organization. All examples can be found within the Tidyverse package. If you load the tidyverse library, and type in table1 for example, the first table will show up.

Tidy Rules

- Three interrelated rules 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.



Which Dataset is Tidy?

```
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
#> # 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
```

```
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
```

Why is Tidy Data Useful?

- It helps to pick a consistent way to store data.
- R is a vectorized language, so when you put variables in columns, R's at its best.

How to Create Tidy Data

- Most often, data does not start out as tidy because it is organized in a format that's easy for collection and entry. Thus we must tidy our data.

- Two important functions to learn:

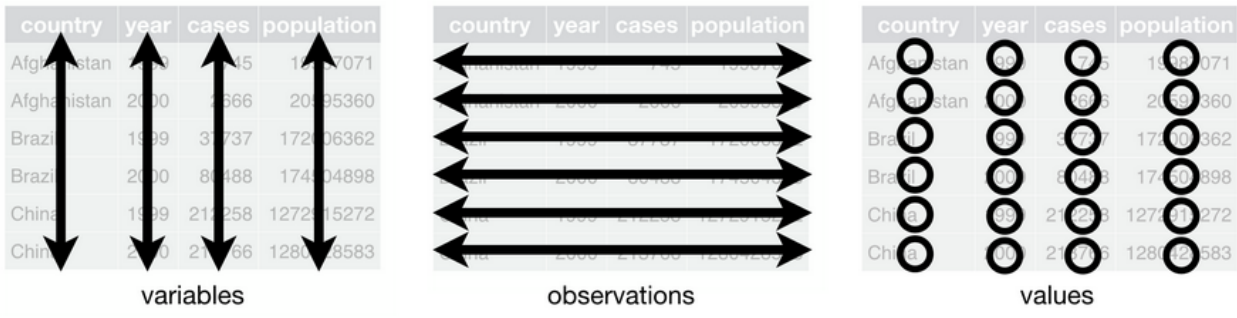
1. **`pivot_longer()`** - when some of the column names are not the names of variables but the *values* of the variables.
   - Wide to Long data
2. **`pivot_wider()`** - when an observation is scattered across multiple rows.
   - Long to Wide data

pivot_longer()

Column names are not the names of variables but the *values* of the variables. *Here for example, 1999 and 2000 are both values of the variable year*

```
table4a
```

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



pivot_longer()

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

- In the tidyverse, you can combine multiple operations with the "pipe", or %>%.
- This makes your code clean and more human-readable if you translate %>% to "then".
  - The code above would read, "take `table4a`, then pivot the columns `1999` and `2000` into one longer column named `year`, and its values should be called `cases`."

pivot_longer()

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

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

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

The `names_to` is the name of the new column that will form from multiple old ones.

The `values_to` are the observations that will fill this new column.

pivot_longer()

This is what just happened. Wide to long data.

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

pivot_wider()

Observation is scattered across multiple rows. *Here for example, data from Afghanistan in 1999 is in multiple rows*

```
print(table2, n = 8, width = Inf)
```

```
## # 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
## 7 Brazil       2000 cases           80488
## 8 Brazil       2000 population 174504898
## # ... with 4 more rows
```

pivot_wider()

```
## # 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
## 7 Brazil       2000 cases          80488
## 8 Brazil       2000 population 174504898
## # ... with 4 more rows
```

```
pivot_wider(table2, names_from = type, values_from = count) %>%
  print(n = 4, width = Inf)
```

```
## # 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
## # ... with 2 more rows
```

The `names_from` is the column with the variables that will spread to 2 columns.

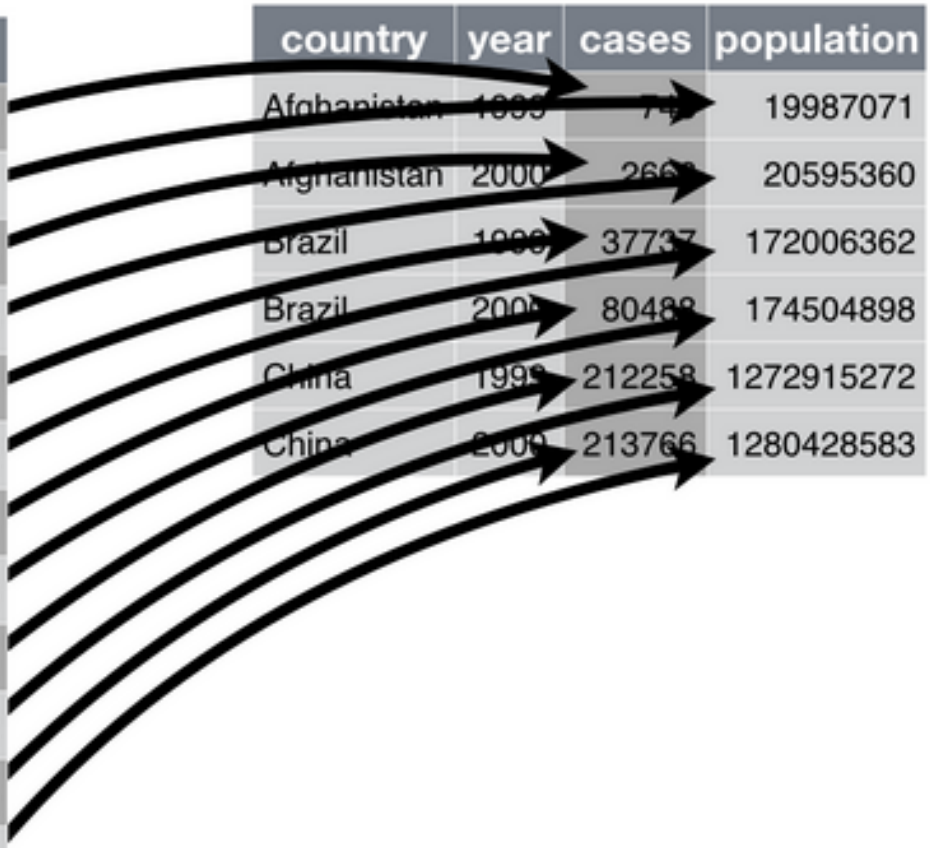
The `values_from` is the column of values that will spread into those 2 columns.

pivot_wider()

This is what just happened. Long to wide data

country	year	key	value
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

table2



separate()

Pulls apart one column into multiple columns wherever a separator appears.

```
## # 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
## # ... with 2 more rows
```

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

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

separate()

If you look carefully, the resulting tibble from the previous set of operations resulted in two new columns that were both *characters* because the original column was a character before the separation. Therefore you should have the function convert to a better type of data using `convert = TRUE`.

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

```
## # 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
```

unite()

Turns two columns into one. The default for `sep` is an underscore (_), so if you want something different you must specify.

```
## # 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
## # ... with 2 more rows
```

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

```
## # 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

Manipulating your data brings up the importance of missing values. These can either be...

1. **Explicit** - flagged with an `NA`
2. **Implicit** - not present in the data

```
frogs <- tibble(
  year       = c(2019, 2019, 2019, 2019, 2018, 2018, 2018),
  individual  = c(1,    2,    3,    4,    2,    3,    4),
  mass       = c(2.88, 3.51, 1.95,   NA, 2.72, 2.17, 3.32))
```

```
## # A tibble: 7 x 3
##    year individual  mass
##   <dbl>      <dbl> <dbl>
## 1  2019          1  2.88
## 2  2019          2  3.51
## 3  2019          3  1.95
## 4  2019          4 NA
## 5  2018          2  2.72
## 6  2018          3  2.17
## 7  2018          4  3.32
```

Missing Values

In our case here, you can make implicit missing values become explicit by pivoting the years into the columns

```
frogs %>%
  pivot_wider(names_from = year, values_from = mass)
```

```
## # A tibble: 4 x 3
##   individual `2019` `2018`
##        <dbl>  <dbl>  <dbl>
## 1          1   2.88  NA
## 2          2   3.51   2.72
## 3          3   1.95   2.17
## 4          4  NA      3.32
```

Missing Values

Or, if these missing values are not important, you can turn these explicit values into implicit ones by using `values_drop_na = TRUE`.

```
frogs %>%
  pivot_wider(names_from = year, values_from = mass) %>%
  pivot_longer(names_to = "year", values_to = "mass", '2018':'2019',
               values_drop_na =  TRUE)
```

```
## # A tibble: 6 x 3
##   individual year   mass
##        <dbl> <chr> <dbl>
## 1          1 2019   2.88
## 2          2 2018   2.72
## 3          2 2019   3.51
## 4          3 2018   2.17
## 5          3 2019   1.95
## 6          4 2018   3.32
```

FYI this is another way to write: `pivot_longer( c('2018','2019'),names_to = "year", values_to = "mass", '2018':'2019', values_drop_na =  TRUE)`

Missing Values

You can also make missing values explicit with `complete()`

```
frogs %>%
  complete(year, individual)
```

```
## # A tibble: 8 x 3
##    year individual  mass
##   <dbl>      <dbl> <dbl>
## 1  2018          1 NA
## 2  2018          2  2.72
## 3  2018          3  2.17
## 4  2018          4  3.32
## 5  2019          1  2.88
## 6  2019          2  3.51
## 7  2019          3  1.95
## 8  2019          4 NA
```