Introduction to R and the tidyverse

– Wrangling part 2: join & reshape –

Paolo Crosetto

Todays topics

> today we will deal with `two` topics:

1. `joining` data from different tables
2. `tidying` data
    1. reshaping
    2. uniting and separating

1. Joining datasets

data scattered around

> you *never* have all the data you need in *one* dataset

- it is usually `scattered` around several datasets
- that might or might not be linked / linkable
  - e.g. data from different `sources` (INSEE and Eurostat)
  - e.g. `computations` to be merged back

oh no! nycflights13 again

```
 1 library(nycflights13)
 2 planes <- nycflights13::planes
 3 planes
```

```
# A tibble: 3,322 × 9
   tailnum  year type              manufacturer model engines seats speed
engine
   <chr>   <int> <chr>             <chr>        <chr>   <int> <int> <int>
<chr>
 1 N10156   2004 Fixed wing multi… EMBRAER      EMB-…       2    55    NA
Turbo…
 2 N102UW   1998 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
 3 N103US   1999 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
 4 N104UW   1999 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
 5 N10575   2002 Fixed wing multi… EMBRAER      EMB-…       2    55    NA
```

oh no! nycflights13 again

```
 1 airports <- nycflights13::airports
 2 airports
```

```
# A tibble: 1,458 × 8
   faa   name                            lat    lon   alt    tz dst   tzone
   <chr> <chr>                         <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
 1 04G   Lansdowne Airport              41.1  -80.6  1044    -5 A
America/…
 2 06A   Moton Field Municipal Airport  32.5  -85.7   264    -6 A
America/…
 3 06C   Schaumburg Regional            42.0  -88.1   801    -6 A
America/…
 4 06N   Randall Airport                41.4  -74.4   523    -5 A
America/…
 5 09J   Jekyll Island Airport          31.1  -81.4    11    -5 A
America/…
 6 0A9   Elizabethton Municipal Airport 36.4  -82.2  1593    -5 A
```

oh no! nycflights13 again

```
1 flights <- nycflights13::flights
2 flights
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
```

Inspect the datasets

> what do these dataset contain?

- what variables do they have in common?
- do they have some unique identifier (`key`)?
- how are these related to one another?

the datasets

- `planes`: info on each plane
  - model, type, date of construction…
- `airports`: info on each airport
  - FAA code, location, latitude, longitude…
- `flights`: info on each flight that left/landed in NYC
  - we know this one already

`join`ing: example

> problem: do **newer** planes fly the **longest** routes from NYC?

- to answer this, you need to combine data from `two` sources:
- `flights` to get the **route's length** in terms of miles
- `planes` to get the **date** the plane was first operational

> how do you *join* the two data frames?

joining: key

- **first** you need to find a *unique identifier* for your data: a **key**
- unique identifiers must be (euh…) *unique* in the `whole` dataset
- in order to find them, either you **`use your intuition`**
- or you **`check`**

finding the right key: `count()`

> one way to check is to count and then filter

```
1 planes %>% count(tailnum) %>% filter(n>1)
```

```
# A tibble: 0 × 2
# i 2 variables: tailnum <chr>, n <int>
```

- `count(var)`: how many times each element of `var` appears as a new variable `n`
- `filter(n>1)`: check if any value appears more than once

finding the right key: `group_by()`

```
 1  planes %>% group_by(tailnum)
```

```
# A tibble: 3,322 × 9
# Groups:   tailnum [3,322]
   tailnum  year type              manufacturer model engines seats speed
engine
   <chr>   <int> <chr>             <chr>        <chr>   <int> <int> <int>
<chr>
 1 N10156   2004 Fixed wing multi… EMBRAER      EMB-…       2    55    NA
Turbo…
 2 N102UW   1998 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
 3 N103US   1999 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
 4 N104UW   1999 Fixed wing multi… AIRBUS INDU… A320…       2   182    NA
Turbo…
```

- `group_by()`: creates a group for each value of *var*.
- If $\#groups = \#observations$, then key is unique

what is joining?

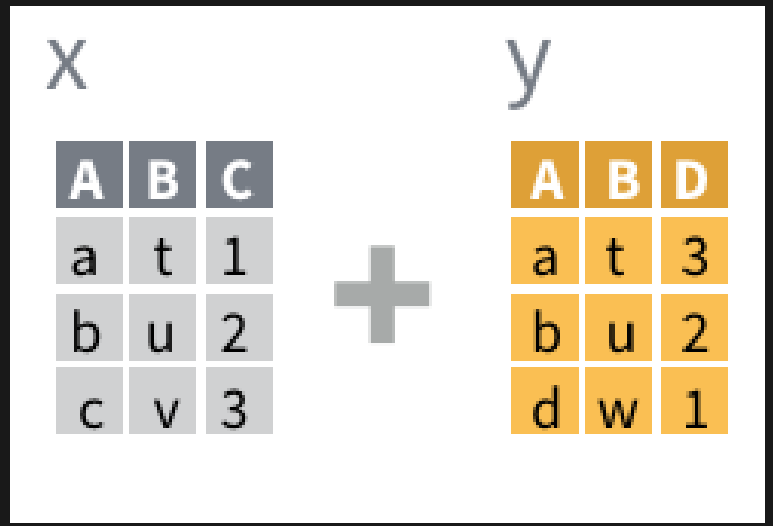
> joining always **combines** data from ***two*** tables into ***one***

- syntax: `join(left, right, by = "key")`
- `left` and `right` two data frames
- `key` the unique identifier in **both** data frames

> different `join()` functions make different assumptions about what to do of the data that are **NOT matched**

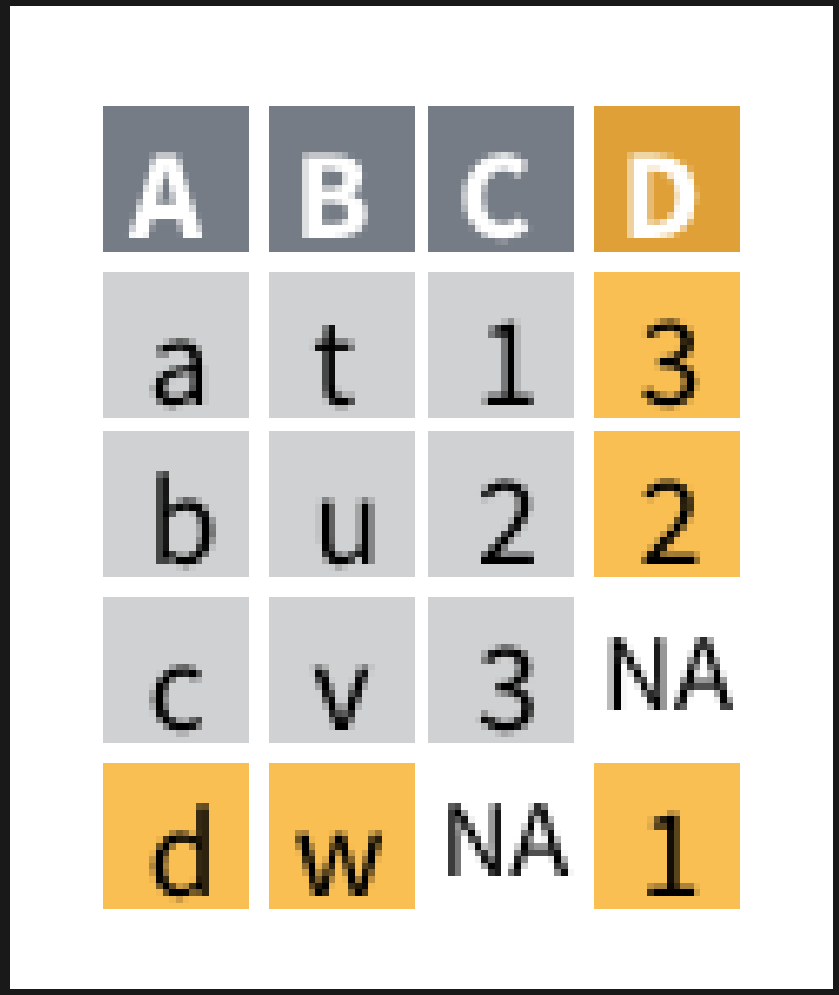
the `*_join()` family of functions

> with a **key** you can use the `*_join()` family of functions



- some **overlapping**, some **new** info on the two tables
- column `D` only in `Y`
- row '`c`' only in dataset `X`; row '`d`' only in dataset `Y`

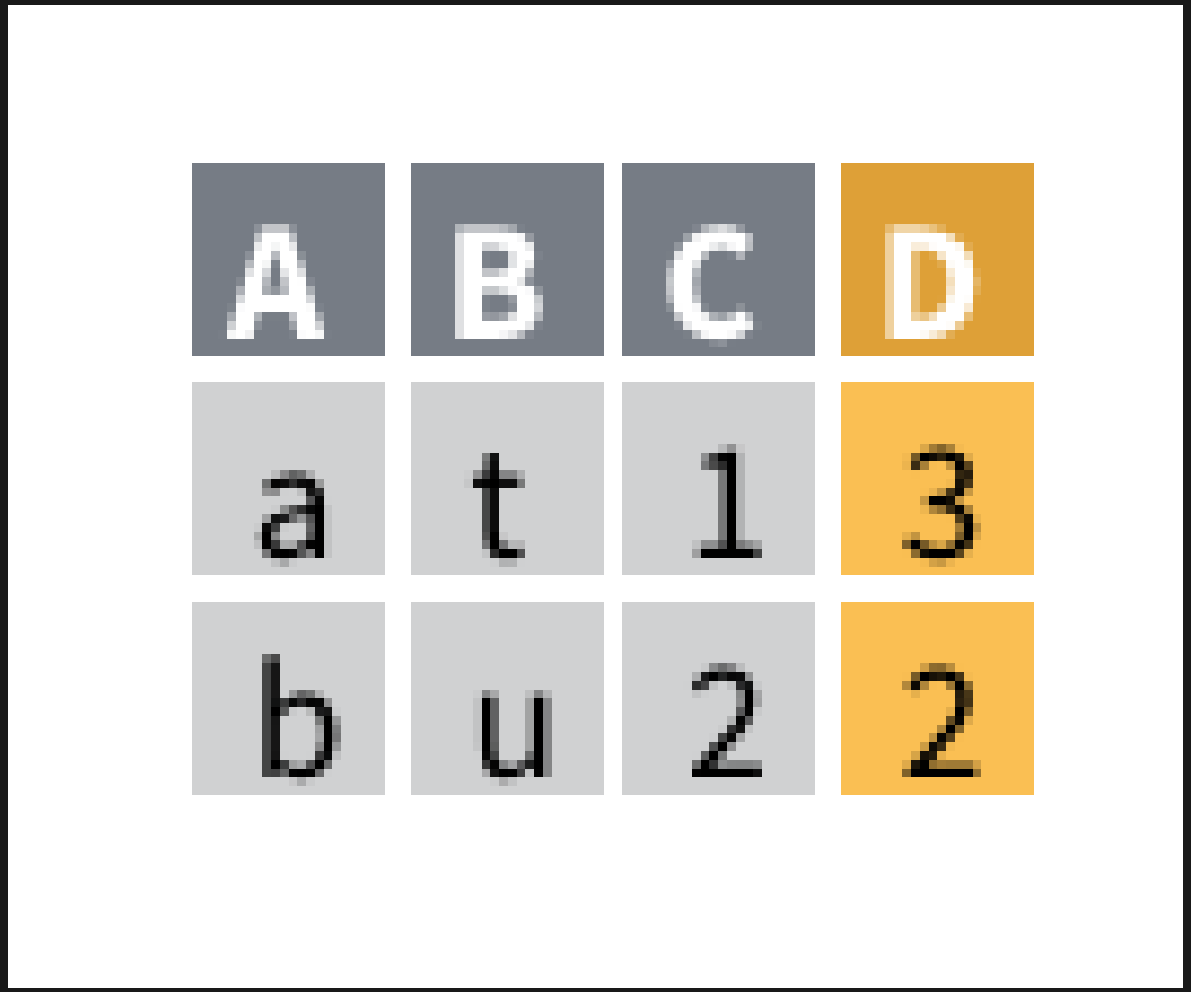
full_join()



A	B	C	D
a	t	1	3
b	u	2	2
c	v	3	NA
d	w	NA	1

`full_join()` keeps everything, adds `NA`

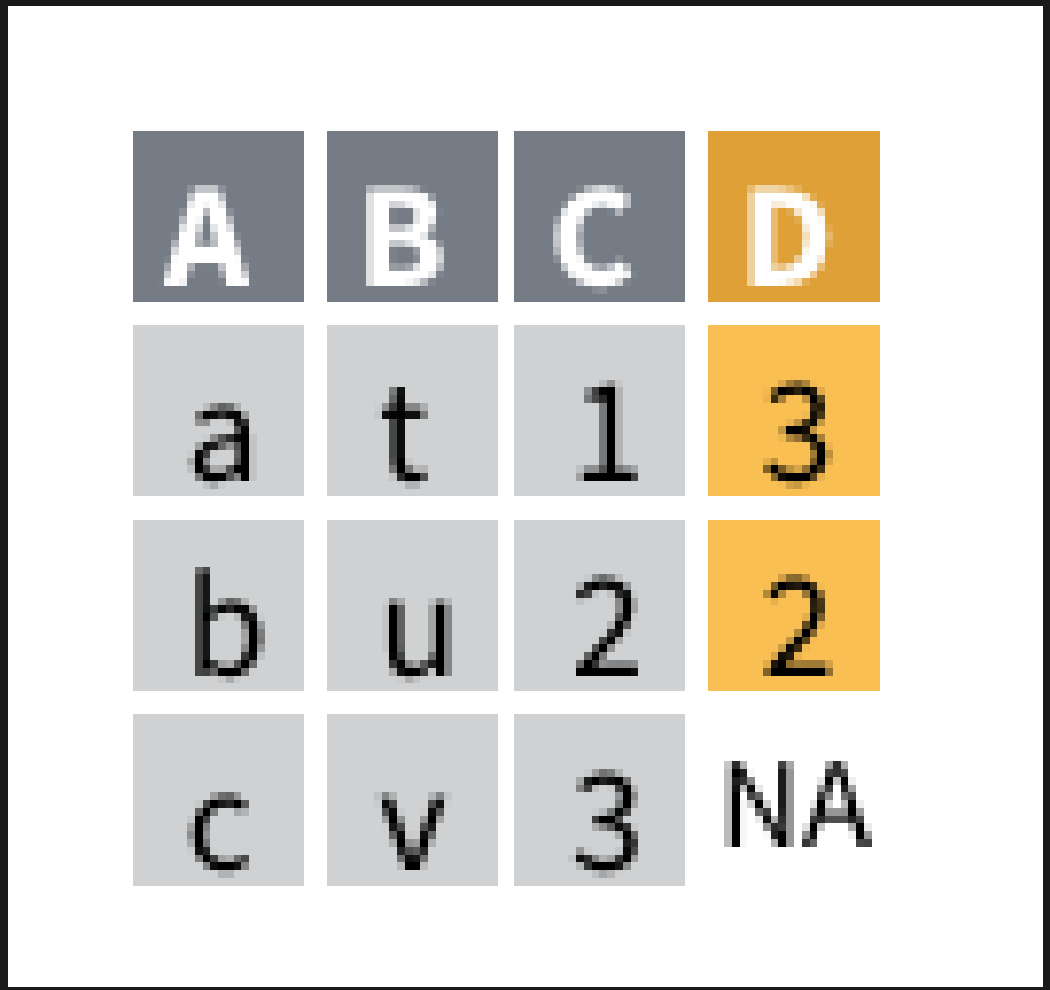
inner_join()



A	B	C	D
a	t	1	3
b	u	2	2

> `inner_join()` keeps only matched data (!! **data loss** !!)

`left_join()`



A	B	C	D
a	t	1	3
b	u	2	2
c	v	3	NA

`left_join()` keeps *all* keys in the **left** data frame

the default `left_join()`

> `left_join()` is the default because you usually add *some* variable to a large dataset

- in our case: **do newer planes fly the longest routes from NYC?**
- most information is on the `flights` dataset
- we need only the **year built** from the `planes` dataset

answering our question: joining

```r
distance <- flights %>% select(tailnum, distance)
yearbuilt <- planes %>% select(tailnum, year)
answer <- left_join(distance, yearbuilt, by = "tailnum")
answer
```

```
# A tibble: 336,776 × 3
   tailnum distance  year
   <chr>      <dbl> <int>
 1 N14228      1400  1999
 2 N24211      1416  1998
 3 N619AA      1089  1990
 4 N804JB      1576  2012
 5 N668DN       762  1991
 6 N39463       719  2012
 7 N516JB      1065  2000
 8 N829AS       229  1998
 9 N593JB       944  2004
10 N3ALAA       733    NA
# i 336,766 more rows
```

answering our question: answer

- no connection between plane's age and the length of the flight

```
answer %>% group_by(year) %>%
  summarise(dist = mean(distance, na.rm = TRUE)) %>%
  ggplot(aes(x = year, y = dist))+geom_point()+
  geom_smooth(method = "lm")
```

Tidy data

messy data -> tidy data

> "*Happy families are all alike; every unhappy family is unhappy in its own way.*" —- Leo Tolstoy

- the data we have worked with so far are all `tidy`
- this is *not* the case in real life
- we need to be able to *format data in a convenient way*

a simple dataset in four versions

```
  1  table1
```

```
# A tibble: 6 × 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
```

a simple dataset in four versions

```
  1  table2
```

```
# A tibble: 12 × 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 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
 9 China        1999 cases           212258
10 China        1999 population 1272915272
11 China        2000 cases           213766
```

a simple dataset in four versions

```
  1  table3
```

```
# A tibble: 6 × 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
```

a simple dataset in four versions

```
  1 table4a  #cases
```

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

```
  1 table4b  #population
```

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

`tidy` *vs* `untidy` data

> `tidy` data:

- each variable has its own column
- each observation has its own row
- each value has its own cell

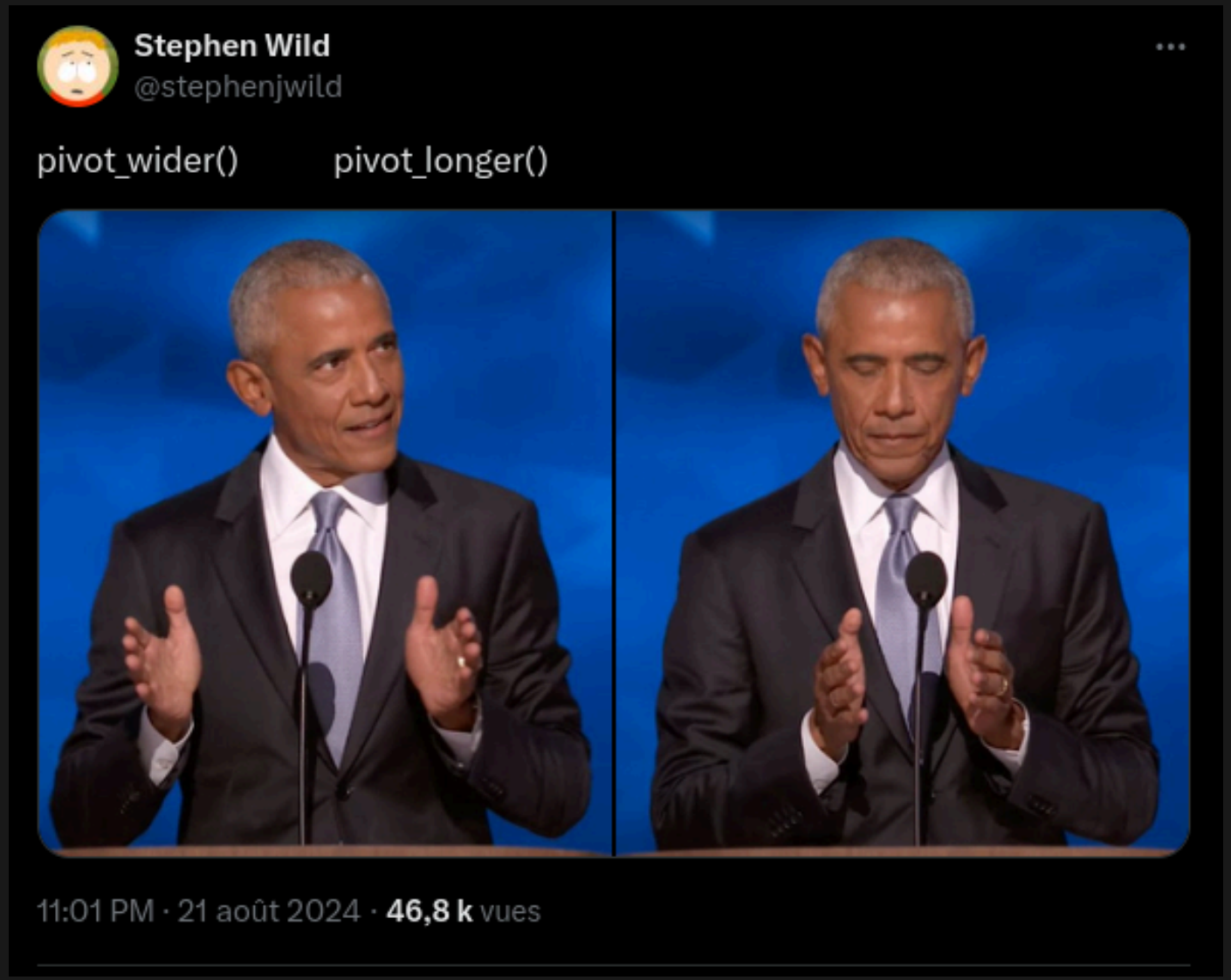
> take a look at the tables.

- what is an observation? a variable?
- do you see problems in the tables?

tidy data: the `tidyr` package

- `tidyr` is part of the tidyverse
- it is automatically loaded with `library(tidyverse)`
- `tidyr` provides 4 main verbs
- `pivot_longer` vs. `pivot_wider`
- `separate` vs. `unite`

2. Reshaping


Stephen Wild
@stephenjwild
pivot_wider()
pivot_longer()
11:01 PM · 21 août 2024 · 46,8 k vues

from wide to long: `pivot_longer`

> sometimes variables are in the column names: bad!

```
  1  table4a
```

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

- `year` is a variable but it is on the column names
- the content in the cells (`cases`) has no variable name

`pivot`ing to longer

> we need to reshape the data so that *year* becomes a variable and *1999* and *2000* become values.

- `pivot_longer(vars, names_to, values_to)`
- `vars` is the stuff we want to move around
- `names_to` is the (new) name to be given to the (new) column that will store the (former) variable names
- `values_to` is the (new) name to be given to the (new) column that will store the values that were spread over several variables

pivoting to longer

> what if we provide just the columns?

```
 1  table4a %>% pivot_longer(!country)
```

```
# A tibble: 6 × 3
  country     name   value
  <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
```

- it works: it correctly guesses.
- but **do not** trust this – automatic, default stuff messes your data without you noticing.

pivoting to longer

> what if we provide full arguments?

```
1 table4a %>% pivot_longer(!country, names_to = "year", values_to = "val")
```

```
# A tibble: 6 × 3
  country     year     val
  <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
```

pivoting to wider

Let's look at a messy version of table1.

```
 1 table2
```

```
# A tibble: 12 × 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 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
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
```

`pivot`ing to wider

> we need to reshape the data from *long* to *wide*, so that *type* gets split into the variables *cases* and *population* and *count* values get assigned to the proper column.

- `pivot_wider(vars, names_from =, values_from =)`
- `names_from` is the (existing) name of the column that contains variable names
- `values_from` is the (existing) name of the variable that contains values of the (to be created) variables

pivoting to wider

```
1 table2 %>% pivot_wider(names_from = type, values_from = count)
```

```
# A tibble: 6 × 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
```

`pivot_longer` <> `pivot_wider`

> if you `pivot_longer` a data frame then you `pivot_wider` it, you are back to the original df.

pivot_longer <> pivot_wider

- step 1: mess up the data

```
 1  table1 %>% pivot_longer(!country & !year)
```

```
# A tibble: 12 × 4
   country      year name           value
   <chr>       <dbl> <chr>          <dbl>
 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
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
```

pivot_longer <> pivot_wider

- step 2: get back in shape

```
  1  table1 %>% pivot_longer(!country & !year) %>% pivot_wider(names_from = name
```

```
# A tibble: 6 × 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
```

3. Unite & separate

`separate()`: one to many

> what is wrong with this table?

```
  1  table3
```

```
# A tibble: 6 × 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
```

`separate()`

> the variable *rate* contains two informations: number of cases and population

- we need to `separate` the variable *into* two (in this case) variables

```
 1 separate(table3, col = rate, into = c("cases", "population"))
```

```
# A tibble: 6 × 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
```

separate()

> `separate()` correctly guessed that the point to separate was "/"

- better to spell out the actual separator

```
 1 separate(table3, col = rate, into = c("cases", "population"), sep = "7")
```

```
# A tibble: 6 × 4
  country      year cases           population
  <chr>       <dbl> <chr>           <chr>
1 Afghanistan  1999 ""              "45/1998"
2 Afghanistan  2000 "2666/20595360"  <NA>
3 Brazil       1999 "3"             ""
4 Brazil       2000 "80488/1"       "4504898"
5 China        1999 "212258/12"     "29152"
6 China        2000 "213"           "66/1280428583"
```

`separate()`

> `separate()` keeps the variables as *characters*

- this is safe: doesn't make assumptions
- but sometimes it is best to have it create `int` or `dbl` variables

```
 1 separate(table3, col = rate, into = c("cases", "population"), convert = TRU
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <dbl>  <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()`: many to one

```
  1  table5
```

```
# A tibble: 6 × 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
```

unite()

> the complementary verb to `separate()` is `unite()`

```
1 unite(table5, year, century, year)
```

```
# A tibble: 6 × 3
  country     year  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
```

- by deault `unite()` uses `_` as a separator

unite()

```
  1 unite(table5, year, century, year, sep = "")
```

```
# A tibble: 6 × 3
  country     year  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
```

`unite()`

- `unite()` creates *string* variables
- if you want to `unite()` in some mathematical way (e.g. sum) use `mutate()`

4. Exercises

Have a look at the Exercise folder, and let’s have fun!