Data Wrangling

Erasmus Q-Intelligence B.V.

Data Science and Business Analytics
Programming

Introduction

Introduction

Data Wrangling is needed since

$\longrightarrow$ Data is often in a messy format

$\longrightarrow$ Data is often not readily usable for analysis and plotting

Data Wrangling often takes a considerable amount of time!

Content

1 Introduction
2 Software requirements
3 Data transformation with `dplyr`
  - `filter()`
  - `arrange()`
  - `select()`
  - `mutate()`
  - `summarise()` and `group_by()`
4 (Intermezzo) Pipes
5 Data transformation with `dplyr` (part 2)
  - `across()`
6 Joining `data.frames`
7 Data transformation with `tidyr`
  - `pivot_longer()` and `pivot_wider()`
8 Conclusion

References to Online Book

- Chapter 3
- Chapter 5
- Chapter 19

Software requirements

Software requirements

```
R> install.packages("nycflights13")
```

```
R> library("nycflights13")
R> library("tidyverse")
```

- ⟶ Note that `dplyr` and `tidyr` are loaded with `tidyverse` and contain the functions we primarily use in this lecture
- ⟶ Note that two functions from `dplyr` - `filter` and `lag` - mask functions with the same name from the `stats`-package
  - ⟶ use the full name to still use these functions: `stats::filter` and `stats::lag`

Data set

Flight information of airports in New York in 2013

```
R> data("flights")
R> ?flights
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <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
# i 336,771 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

Data transformation with `dplyr`

Five basic functions

The 5 most common functions in the `dplyr`-package:

⟶ `filter()`: filter observations (rows) based on content

⟶ `arrange()`: arrange observations (rows)

⟶ `select()`: select variables (columns) based on names

⟶ `mutate()`: mutate or add new variables (columns)

⟶ `summarise()`: make a summary of the data

Combined with `group_by()`, these functions can also be evaluated on groups in the data

⟶ For example, find the average `dep_delay` for each day in the data set (data should be grouped by `day` and `month`)

Five basic functions (2)

Every of these functions works the same

$\longrightarrow$ the first argument is the data set

$\longrightarrow$ the next arguments describe what you want to do with the data

$\longrightarrow$ the result is a new, transformed `data.frame`

filter()

Example: *Find all flights on March 13th*

```
R> flights_mar13 <- filter(flights, month == 3 & day == 13)
```

```
# A tibble: 974 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     3    13      103           2355        68      457
2  2013     3    13      458            500        -2      648
3  2013     3    13      515            515         0      805
4  2013     3    13      525            530        -5      821
5  2013     3    13      541            545        -4      920
# i 969 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

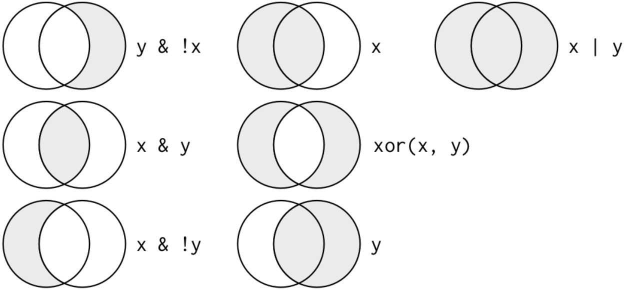
`filter()` (2)

On the previous slide, we used the *Boolean*-notation `&`

$\longrightarrow$ `&`: 'and' - both condition should be `TRUE` to be selected

$\longrightarrow$ `|`: 'or' - any condition should be `TRUE` to be selected

$\longrightarrow$ `!`: 'not' - condition should be `FALSE` to be selected



filter() (3)

If a value should be equal to a value in a given series, use %in%

Example: *Find all flights in the 4th quarter*

```
R> flights_q4 <- filter(flights, month %in% c(10:12))
```

In a first data analysis, you often want to find all observations with missings:

```
R> flights_no_dep_time <- filter(flights, is.na(dep_time))
```

Exercises

Download and open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.1

arrange()

Example: *Arrange* `flights` *by* `dep_delay`

```
R> flights_arranged <- arrange(flights, dep_delay)
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013    12     7     2040           2123       -43       40
2  2013     2     3     2022           2055       -33     2240
3  2013    11    10     1408           1440       -32     1549
4  2013     1    11     1900           1930       -30     2233
5  2013     1    29     1703           1730       -27     1947
# i 336,771 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

`arrange()` (2)

If you want to arrange in descending order, use `desc(dep_delay)`

Example: *Arrange* `flights` *by* `dep_delay`*, in descending order*

```
R> flights_arranged <- arrange(flights, desc(dep_delay))
```

⟶ Missing values (`NA`) are always put last

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.2

select()

Example: *Select columns* `carrier`, `origin`, `dest`, `dep_delay`, `arr_delay` *from* `flights`

```
R> flights_selection <- select(flights, carrier, origin, dest,
+                                    dep_delay, arr_delay)
```

```
# A tibble: 336,776 x 5
  carrier origin dest  dep_delay arr_delay
  <chr>   <chr>  <chr>     <dbl>     <dbl>
1 UA      EWR    IAH           2        11
2 UA      LGA    IAH           4        20
3 AA      JFK    MIA           2        33
4 B6      JFK    BQN          -1       -18
5 DL      LGA    ATL          -6       -25
# i 336,771 more rows
```

select() (2)

You can select a range of columns by using ‘:‘

```
R> flights_selection <- select(flights, carrier:dest)
```

Or specify which you *don't* want to select with ‘-‘

```
R> flights_selection <- select(flights, -(year:day))
```

Columns are reodered by the order you set in `select`

```
R> flights_reordered <- select(flights, flight, tailnum, carrier,
+                                origin, dest, everything())
```

`everything()` selects all columns, but columns will not be duplicated!

`select()` (3)

Helpful functions for column-selection:

- ⟶ `starts_with('abc')`: select all columns starting with `abc`
- ⟶ `ends_with('abc')`: select all columns ending with `abc`
- ⟶ `contains('abc')`: select all columns containing `abc`
- ⟶ `one_of('year', 'date')`: select all columns with these names, and does not crash if (e.g.) `'date'` does not exist

(for all 'select helpers', see `?dplyr_tidy_select`)

```
R> flights_time <- select(flights, contains('time'))
```

```
# A tibble: 336,776 x 6
  dep_time sched_dep_time arr_time sched_arr_time air_time
     <int>          <int>    <int>          <int>    <dbl>
1      517            515      830            819      227
2      533            529      850            830      227
# i 336,774 more rows
# i 1 more variable: time_hour <dttm>
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.3

mutate()

Example: *Create a column giving the gain in delay, and a column giving the gain per minute*

```
R> flights_selection <- select(flights,
+                                year:day, ends_with('delay'),
+                                air_time)
R> flights_gain <- mutate(flights_selection,
+                           gain = dep_delay - arr_delay,
+                           gain_per_minute = gain/air_time)
```

```
# A tibble: 336,776 x 8
   year month   day dep_delay arr_delay air_time  gain
  <int> <int> <int>     <dbl>     <dbl>    <dbl> <dbl>
1  2013     1     1         2        11      227    -9
2  2013     1     1         4        20      227   -16
3  2013     1     1         2        33      160   -31
# i 336,773 more rows
# i 1 more variable: gain_per_minute <dbl>
```

`mutate()` (2)

- ⟶ the second mutation can use the first mutation
- ⟶ any function can be used, as long as the function results in a vector with the same length as the original `data.frame`

```
R> flights_gain <- mutate(flights_gain,
+                         gain_per_hour = gain_per_minute*60,
+                         cum_gain = cummean(gain_per_hour))
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.4

summarise()

Example: *Get the mean of* `dep_delay`

```
R> flights_summary <- summarise(flights,
+                               delay = mean(dep_delay,
+                                            na.rm = TRUE))
```

```
# A tibble: 1 x 1
  delay
  <dbl>
1  12.6
```

Not so interesting. Better: Get the mean of `dep_delay` by day

group_by()

Example: *Get the mean of* `dep_delay` *by day*

```
R> flights_grouped <- group_by(flights, month, day)
R> flights_summary <- summarise(flights_grouped,
+                                  delay = mean(dep_delay,
+                                               na.rm = TRUE))
```

```
# A tibble: 365 x 3
# Groups:   month [12]
  month   day delay
  <int> <int> <dbl>
1     1     1 11.5
2     1     2 13.9
3     1     3 11.0
4     1     4  8.95
5     1     5  5.73
# i 360 more rows
```

summarise() **and** group_by()

Helpful functions for summarise() (by group using group_by()):

$\longrightarrow$ mean(), median(), sd(): average, median or standard deviation of observations

$\longrightarrow$ quantile(): quantile of distribution

$\longrightarrow$ first(), nth(), last(): select specific observation

$\longrightarrow$ n(), n_distinct(), sum(!is.na()): number of observations, of distinct observations and non-missings

summarise() and group_by() (2)

```
R> flights_summary <- summarise(flights_grouped,
+                                 delay = mean(dep_delay,
+                                              na.rm = TRUE),
+                                 first_delay = first(dep_delay),
+                                 nr_fligths = n())
```

```
# A tibble: 365 x 5
# Groups:   month [12]
  month   day delay first_delay nr_fligths
  <int> <int> <dbl>       <dbl>      <int>
1     1     1 11.5            2        842
2     1     2 13.9           43        943
3     1     3 11.0           33        914
4     1     4  8.95          26        915
5     1     5  5.73          15        720
# i 360 more rows
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.5

`group_by()`, `filter()` **and** `mutate()`

`mutate()` and `filter()` can also be used together with `group_by`

Example: *Find, for each day, the 9 most delayed flights*

```
R> flights_by_day <- group_by(flights, month, day)
R> flights_most_delay <- filter(flights_by_day,
+                                rank(desc(arr_delay)) < 10)
```

(Why is the number of observations not equal to 365 x 9?)

group_by(), filter() and mutate()

mutate() and filter() can also be used together with group_by

Example: *Calculate, by destination, how much a flight adds to the total delay*

```
R> flights_by_dest <- group_by(flights, dest)
R> flights_delays <- filter(flights_by_dest, arr_delay > 0)
R> flights_prop_delay <- mutate(flights_delays,
+                                 prop_delay = arr_delay /
+                                   sum(arr_delay, na.rm = TRUE))
```

sum(arr_delay, na.rm = TRUE) is the sum over all positive delays, <u>for that destination</u>

(Intermezzo) Pipes

Pipes

Having to save every step to a `data.frame` in your environment is not very efficient

$\longrightarrow$ 'In between'-results are not of interest

Therefore, use pipes (%>%) instead

```
R> flights_summary <- flights %>%
+    group_by(month, day) %>%
+    summarise(delay = mean(dep_delay, na.rm = TRUE),
+              first_delay = first(dep_delay),
+              nr_fligths = n())
```

Pipes (2)

You do not have to repeat the data you work with and do not have to save every 'In between'-result!

$\longrightarrow$ Especially useful if many transformations (`filter()`, `select()`, `mutate()`, `summarise()`) have to be done

Example: *What does this code do?*

```
R> flights_gains <- flights %>%
+    filter(month == 3 & day == 13) %>%
+    select(carrier, origin, dest, dep_delay, arr_delay) %>%
+    mutate(gain = dep_delay - arr_delay) %>%
+    group_by(dest) %>%
+    summarise(mean_gain = mean(gain, na.rm = TRUE),
+              max_gain = max(gain, na.rm = TRUE),
+              nr_flights = n())
```

Pipes (3)

```
# A tibble: 83 x 4
   dest  mean_gain max_gain nr_flights
   <chr>     <dbl>    <dbl>      <int>
 1 ALB        8.5        11          2
 2 ATL        3.06       21         50
 3 AUS       14.8        31         11
 4 BDL       13          22          2
 5 BHM       -1          -1          1
 6 BNA       12.9        21         18
 7 BOS       10.8        28         48
 8 BQN       -3.67        3          3
 9 BTV        9          20          9
10 BUF        3.36       16         12
# i 73 more rows
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.6 and 1.7

Data transformation with `dplyr` (part 2)

across()

Applying the same function to a set of columns of the data frame

```
R> flights_no_across <- summarise(flights_grouped,
+     dep_delay = mean(dep_delay, na.rm = TRUE),
+     arr_delay = mean(arr_delay, na.rm = TRUE),
+     air_time = mean(air_time, na.rm = TRUE))
```

Having to copy the same code is not very efficient

$\longrightarrow$ `across()` to the rescue!

`across()` can be used for `summarise()` and `mutate()`

`across()` (2)

Example: *Find the average for* `dep_delay` , `arr_delay` *and* `air_time`

```
R> flights_across <- summarise(flights_grouped,
+     across(c(dep_delay, arr_delay, air_time),
+            ~mean(., na.rm = TRUE)))
```

`across(.cols, .fns)`

- `.cols` specifies on which columns the function should be executed
- `.fns` specifies which function should be executed on the columns

across() - .cols

For .cols, several inputs can be used

⟶ everything(): apply the function to all columns

⟶ c(): ... to a list of column names

⟶ starts_with(): ... to all columns that start with...

⟶ where(is.numeric): ... to all numeric columns

Example *Find the number of distinct values for all character variables in* flights

```
R> flights_across <- summarise(flights,
+     across(where(is.character), n_distinct))
```

```
R> print(flights_across)
# A tibble: 1 x 4
  carrier tailnum origin  dest
    <int>   <int>  <int> <int>
1      16    4044      3   105
```

across() - .fns

For `.fns`, several inputs can be used

⟶ `mean`: directly the name of the function

⟶ `~mean(., na.rm = TRUE)`: a function if extra input arguments are needed

⟶ `function_name`: your own (list of) function(s)

Example *Find the mean and median, by day, for* `dep_delay` , `arr_delay` *and* `air_time`

```
R> mean_median <- list(mean = ~mean(., na.rm = TRUE),
+         median = ~median(., na.rm = TRUE))
R>
R> flights_across <- flights %>%
+       group_by(month, day) %>%
+       summarise(across(c(dep_delay, arr_delay, air_time),
+                               mean_median))
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercise 1.8

Joining `data.frames`

Relational data

Often, a dataset from a database consists of multiple separate data frames

⟶ To combine these tables, we use `joins` from the `dplyr`-package

In the `nycflights13`-dataset, we have several `data.frames`

⟶ flights

⟶ airlines

⟶ airports

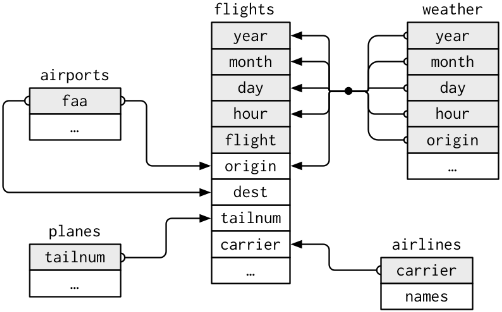
⟶ planes

⟶ weather

Relational data (2)

The relation between the data frames works with a *key*

⟶ A variable (column) which has unique elements and connects two `data.frames`

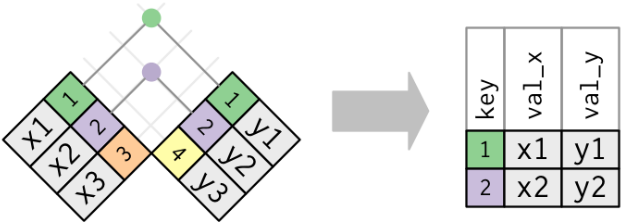


Example: *With* `tailnum` , `planes` *and* `flights` *can be joined.*

Inner-join

Add columns of `data.frame` together based on the `key`

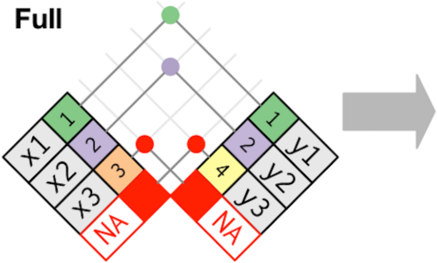
With `inner_join`, keep only rows of which the `key` is in both `data.frames`



Full-join

Add columns of `data.frame` together based on the `key`
With `full_join`, keep all rows from both `data.frames` and join by `key`

$\longrightarrow$ If the value of `key` is not in the other `data.frame`, `NA` is added (Not Available)

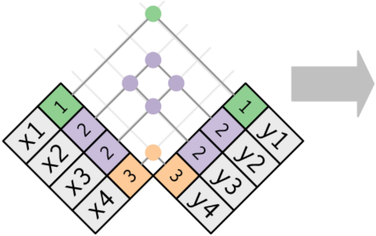


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Left-join

Add columns of `data.frame` together based on the `key`

With `left_join`, keep all rows from the first `data.frame` and match observations with same `key` from the second `data.frame`



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Example

Example: *Join* `flights` *and* `planes` *with* `key` `tailnum`, keeping all observations from `flights`

```
R> data("planes")
R>
R> flights_select <- flights %>%
+    select(year, month, day, tailnum, flight, carrier, origin,
+           dest, dep_time, arr_time)
```

Example (2)

Example: *Join* `flights` *and* `planes` *with* `key` `tailnum`, keeping all observations from `flights`

```
R> print(flights_select, n = 5)
# A tibble: 336,776 x 10
   year month   day tailnum flight carrier origin dest  dep_time
  <int> <int> <int> <chr>    <int> <chr>   <chr>  <chr>    <int>
1  2013     1     1 N14228    1545 UA      EWR    IAH        517
2  2013     1     1 N24211    1714 UA      LGA    IAH        533
3  2013     1     1 N619AA    1141 AA      JFK    MIA        542
4  2013     1     1 N804JB     725 B6      JFK    BQN        544
5  2013     1     1 N668DN     461 DL      LGA    ATL        554
# i 336,771 more rows
# i 1 more variable: arr_time <int>
```

Example (3)

Example: *Join* `flights` *and* `planes` *with* key `tailnum`, keeping all observations from `flights`

```
R> print(planes, n = 5)
# A tibble: 3,322 x 9
  tailnum  year type          manufacturer model engines seats speed
  <chr>   <int> <chr>         <chr>        <chr>   <int> <int> <int>
1 N10156   2004 Fixed wing~ EMBRAER      EMB-~       2    55    NA
2 N102UW   1998 Fixed wing~ AIRBUS INDU~ A320~       2   182    NA
3 N103US   1999 Fixed wing~ AIRBUS INDU~ A320~       2   182    NA
4 N104UW   1999 Fixed wing~ AIRBUS INDU~ A320~       2   182    NA
5 N10575   2002 Fixed wing~ EMBRAER      EMB-~       2    55    NA
# i 3,317 more rows
# i 1 more variable: engine <chr>
```

Example (4)

Example: *Join* `flights` *and* `planes` *with* `key tailnum`, keeping all observations from `flights`

```
R> flights_planes <- flights_select %>%
+    left_join(planes, by = c('tailnum' = 'tailnum'))
```

```
# A tibble: 336,776 x 18
  year.x month   day tailnum flight carrier origin dest  dep_time
   <int> <int> <int> <chr>    <int> <chr>   <chr>  <chr>    <int>
1   2013     1     1 N14228    1545 UA      EWR    IAH        517
2   2013     1     1 N24211    1714 UA      LGA    IAH        533
3   2013     1     1 N619AA    1141 AA      JFK    MIA        542
# i 336,773 more rows
# i 9 more variables: arr_time <int>, year.y <int>, type <chr>,
#   manufacturer <chr>, model <chr>, engines <int>, seats <int>,
#   speed <int>, engine <chr>
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercise 1.9

Data transformation with `tidyr`

Tidy data

Data is tidy if

- Every variable has its own column
- Every observations has its own row
- Every value has its own cell

Tidy data

```
# 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
```

Data is tidy, since all variables and observations have their own columns and rows

Tidy data

```
# 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
4 Afghanistan  2000 population   20595360
5 Brazil       1999 cases           37737
6 Brazil       1999 population 172006362
# i 6 more rows
```

Data is untidy, since variables `cases` and `population` are together in one column

Tidy data

Data is tidy if

- Every variable has its own column
- Every observations has its own row
- Every value has its own cell

Tidy data is

- consistently saved
- easier to manipulate

Often, you will need untidy data to present your data in plots or tables: `tidyr` helps to get your data in the format needed

Gather data spread over columns

Often, a variable is spread over more than one column:

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

Columns `'2019'` and `'2020'` are actually values of the variable `year`

Gather data spread over columns

Use the function `pivot_longer`:

```
R> table_gathered <- table4a %>%
+    pivot_longer(cols = c(`1999`, `2000`),
+                 names_to = 'year',
+                 values_to = 'cases')
```

Function arguments:

- `data` - `data.frame` to be used
- `cols` - columns of data.frame to be gathered
- `names_to` - name of new column created from the column names
- `values_to` - name of new column created from values in columns

Gather data spread over 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
```

Spread data gathered in column

Often, more than one variable is gathered in one column:

```
# 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
4 Afghanistan  2000 population   20595360
5 Brazil       1999 cases           37737
6 Brazil       1999 population  172006362
# i 6 more rows
```

`cases` and `population` are actually separate variables, gathered in column `type`

Spread data gathered in column

Use the function `pivot_wider`:

```
R> table_spread <- table2 %>%
+    pivot_wider(names_from = 'type',
+                values_from = 'count')
```

Function arguments:

- `data` - `data.frame` to be used
- `names_from` - name of column with the new spread column names
- `values_from` - name of column with the values for the new columns

Spread data gathered in column

```
# 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
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.10 and 1.11

Conclusion

Conclusion

- $\longrightarrow$ Data is often in a messy format
- $\longrightarrow$ Data is often not readily usable for analysis and plotting
- $\longrightarrow$ `dplyr` has handy functions to wrangle your data to make it usable
  - $\longrightarrow$ `select` and `filter` to get subsets of the data
  - $\longrightarrow$ `mutate`, `arrange`, `summarise` and `group_by` to manipulate your data

Data Wrangling often takes a considerable amount of time!