Introduction to Data Wrangling I

Summer Institute in Data Science

Rolando J. Acosta

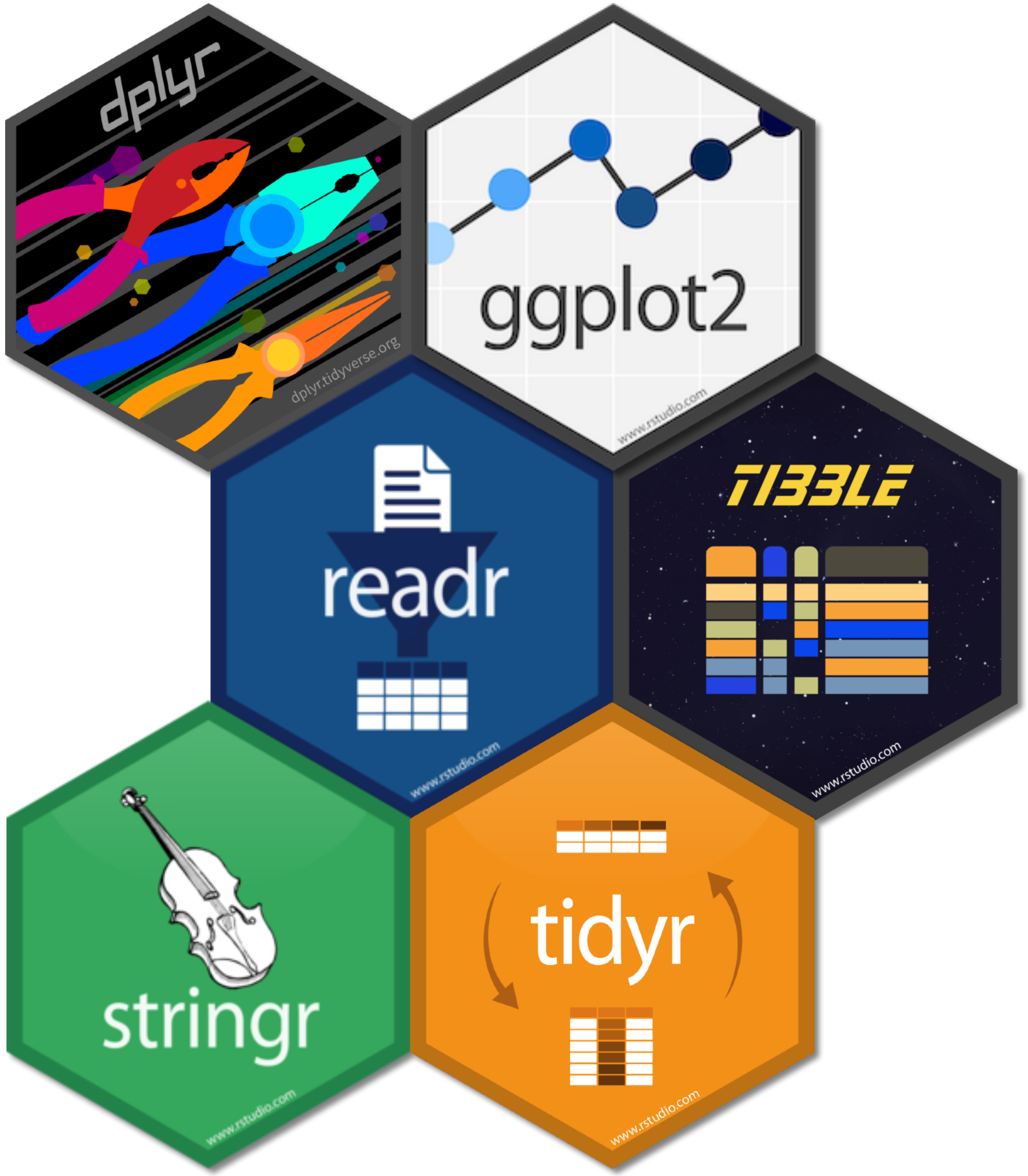
HARVARD
SCHOOL OF PUBLIC HEALTH

June 20, 2022

@RJANunez

Welcome back!

- Last year, we used datasets that were ***tidy***. Rows represented observations and columns variables.
- However, rarely in data science do we get a clean dataset from the get-go
- This week we will learn the basic of data wrangling
- First, let's revisit some concepts we learned last year



Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```r
library(dslabs)
data(murders)
View(murders)
```

state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

Example of tidy format

Transforming data frames

- We can use functions from the package ***dplyr*** to transform data frames:
  - `mutate`
  - `filter`
  - `select`
  - The pipe operator (%>%)
  - `summarize`
  - `group_by`
  - `do`

Adding a column with `mutate()`

- Let's add a column with murder rates to the ***murders*** dataset.
- Syntax for the function `mutate`:

```
mutate(data frame, name = value)
```

- `data frame`: Name of data frame of interest
- `name`: Name of the new column
- `value`: Values that the variable should take

Adding a column with `mutate()`

```
library(dslabs)
library(dplyr)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```

state	abb	region	population	total	rate
Alabama	AL	South	4779736	135	2.8244238
Alaska	AK	West	710231	19	2.6751860
Arizona	AZ	West	6392017	232	3.6295273
Arkansas	AR	South	2915918	93	3.1893901
California	CA	West	37253956	1257	3.3741383
Colorado	CO	West	5029196	65	1.2924531
Connecticut	CT	Northeast	3574097	97	2.7139722
Delaware	DE	South	897934	38	4.2319369

Subsetting with `filter()`

- Say that we want to only show entries with a murder rate lower than or equal to 0.71
- Syntax for the function `filter`:

```
filter(data frame, condition)
```

- `data frame`: Name of data frame of interest
- `condition`: A rule use to subset data

Subsetting with `filter()`

```
filter(murders, rate <= 0.71)
```

state	abb	region	population	total	rate
Hawaii	HI	West	1360301	7	0.5145920
Iowa	IA	North Central	3046355	21	0.6893484
New Hampshire	NH	Northeast	1316470	5	0.3798036
North Dakota	ND	North Central	672591	4	0.5947151
Vermont	VT	Northeast	625741	2	0.3196211

Selecting columns with `select()`

- In this example let's select a few columns from the original dataset and then filter as we did before
- Syntax for the function `select`:

```
select(data frame, columns)
```

- `data frame`: Name of data frame of interest
- `columns`: Name of the columns of interest

Selecting columns with `select()`

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

state	region	rate
Hawaii	West	0.5145920
Iowa	North Central	0.6893484
New Hampshire	Northeast	0.3798036
North Dakota	North Central	0.5947151
Vermont	Northeast	0.3196211

The pipe operator: %>%

- We used the following code in the previous slide:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

- However, we can perform a series of operations by sending the results of one function to another with the pipe operator (%>%)

original data ⟶ `select` ⟶ `filter`

The pipe operator: %>%

- Let's look at few examples:

```
16 %>% sqrt()
```

```
16 %>% sqrt() %>% log2()
```

- The first one yields 4 and the second 2
- Note that the pipe sends values to the first argument, so we can define other arguments as if the first one is defined

```
16 %>% sqrt() %>% log(base = 2)
```

The pipe operator: %>%

- Original code

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

- New code

```
murders %>%
   select(state, region, rate) %>%
   filter(rate <= 0.71)
```

- murders is the first argument to `select` and the result from `select` is the first argument to `filter`

Summarizing data

- An important step in any analysis is summarizing data:
  - mean
  - standard deviation
- Sometimes we can get more informative summaries by first splitting the data by groups and then summarizing
- Let's us introduce to functions to do this:
  - `summarize`
  - `group_by`

Summarizing data

- New dataset: The ***heights*** dataset includes height and sex reported by students in an in-class survey

```
library(dslabs)
library(dplyr)
data(“heights")
```

- The following code computes the mean and standard deviation for females

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(average = mean(height),
             sta_dev = sd(height))
```

sex	height
Male	75.00000
Male	70.00000
Male	68.00000
Male	74.00000
Male	61.00000
Female	65.00000
Female	66.00000
Female	62.00000
Female	66.00000

Sample of ***heights*** dataset

Summarizing data

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(average = mean(height), sta_dev = sd(height))
```

- This yields `average = 64.94` and `sta_dev = 3.76`

- We can compute any number of summary statistics:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(median = median(height), minimum = min(height),
             maximum = max(height))
```

Summarizing data

- Recall that we can get the minimum, median, and maximum statistics by looking at the `0%`, `50%`, and `100%` quantiles:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(range = quantile(height, c(0, 0.5, 1)))
```

- but this return an error!
- This is because we can only call functions that return a single value within `summarize`
- One last example. Let’s compute the murder rate in the US

```
murders %>%
   summarize(rate = sum(total) / sum(population) * 100000)
```

Group and then summarize

- As stated before, is common to first split the data by groups and then provide summaries for each group. Let's compute the mean and standard deviation for males and females separately:

```
heights %>% group_by(sex)
```

Sex	Height
Male	75
Male	70
Male	68
Female	65
Female	66
Female	62

group_by(sex)

Sex	Height
Male	75
Male	70
Male	68

Sex	Height
Female	65
Female	66
Female	62

Group and then summarize

```r
heights %>%
   group_by(sex) %>%
   summarize(average = mean(height), sta_dev = sd(height))
```

Sex	Height
Male	75
Male	70
Male	68
Female	65
Female	66
Female	62

→ group_by(sex)

Sex	Height
Male	75
Male	70
Male	68

→ summarize

Sex	Height
Female	65
Female	66
Female	62

→ summarize

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

```
murders
```

- Start with the dataset that we want to use

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

```
murders %>%
   group_by(region)
```

- Use the pipe operator to “send” the data to the `group_by` function
- Recall that the pipe makes ***murders*** the first argument in `group_by`
- Therefore, the only thing left is to specify which variable to group by

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

```
murders %>%
   group_by(region) %>%
   summarize(median_rate = median(rate))
```

- Finally, use `summarize` to get the median murder rates per region

Sorting data frames

- We can use `arrange` to sort dataframes

```
murders %>%
  arrange(rate)
```

state	abb	region	population	total	rate
Vermont	VT	Northeast	625741	2	0.3196211
New Hampshire	NH	Northeast	1316470	5	0.3798036
Hawaii	HI	West	1360301	7	0.5145920
North Dakota	ND	North Central	672591	4	0.5947151
Iowa	IA	North Central	3046355	21	0.6893484

- To sort in descending order we can use `desc`

```
murders %>%
  arrange(desc(rate))
```

state	abb	region	population	total	rate
District of Columbia	DC	South	601723	99	16.4527532
Louisiana	LA	South	4533372	351	7.7425810
Missouri	MO	North Central	5988927	321	5.3598917
Maryland	MD	South	5773552	293	5.0748655
South Carolina	SC	South	4625364	207	4.4753235

Sorting data frames

- We can also do nested sorting

```
murders %>%
  arrange(region, rate)
```

state	abb	region	population	total	rate
Vermont	VT	Northeast	625741	2	0.3196211
New Hampshire	NH	Northeast	1316470	5	0.3798036
Maine	ME	Northeast	1328361	11	0.8280881
Rhode Island	RI	Northeast	1052567	16	1.5200933
Massachusetts	MA	Northeast	6547629	118	1.8021791

- Finally, if we want to get top *n* observations we can use `top_n`

```
murders %>%
  top_n(5, rate)
```

state	abb	region	population	total	rate
District of Columbia	DC	South	601723	99	16.4527532
Louisiana	LA	South	4533372	351	7.7425810
Missouri	MO	North Central	5988927	321	5.3598917
Maryland	MD	South	5773552	293	5.0748655
South Carolina	SC	South	4625364	207	4.4753235

The do function

- Most R functions do not accept ***tibbles*** nor do they return data frames
- Recall the `quantile` example from before:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(range = quantile(height, c(0, 0.5, 1)))
```

- which yields the following error:

```
Error: expecting result of length one, got : 2
```

- The do function serves as a bridge between R

The do function

- Let's use the do function to get around this

- First, we have to write a function that takes a data frame as an argument and returns a data frame

```
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

- Now we can use the following code:

```
heights %>%
  group_by(sex) %>%
  do(my_summary(.))
```

The do function

- Let's use the do function to get around this
- First, we have to write a function that takes a data frame as an argument and returns a data frame

```
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

- Now we can use the following code:

```
heights %>%
  group_by(sex) %>%
  do(my_summary(.))
```

- Why the dot?

The `do` function

- The ***tibble*** created by `group_by` is piped to `do`
- Within the call to `do`, the name of this tibble is `.` and we want to send it to `my_summary`

Reshaping data

- The first step in the analysis process is *importing the data*.
- Usually, but not always, the next involves *reshaping* the data into a form that facilitates the analysis
- The ***tidyr*** package includes several functions to do this

Reshaping data

```
library(tidyverse)
library(dslabs)

path      <- system.file("extdata", package="dslabs")
filename  <- file.path(path, "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
```

country	1960	1961	1962	1963	1964	1965	1966
Germany	2.41	2.44	2.47	2.49	2.49	2.48	2.44
South Korea	6.16	5.99	5.79	5.57	5.36	5.16	4.99

`pivot_longer`

- `pivot_longer` is a function from the ***tidyr*** package that is useful for turning wide data into tidy data.
- We want to reshape the wide_data dataset so that each row represents a fertility observation, which implies that we need three columns to store the year, country, and observed value.

```
pivot_longer(data frame, cols, names_to, values_to)
```

- `data frame`: Name of data frame of interest
- `cols`: Columns to pivot. These columns will contain the observations of interest
- `names_to`: Column name in the tidy dataset that will contain the column names of the wide dataset
- `values_to`: Column name in the tidy dataset that will contain the observations from the wide dataset

pivot_longer

```
new_tidy_data <- pivot_longer(data      = wide_data,
                              cols      = `1960`:`2015`,
                              names_to  = "year",
                              values_to = "fertility")
```

	country	year	fertility
1	Germany	1960	2.41
2	Germany	1961	2.44
3	Germany	1962	2.47
4	Germany	1963	2.49
5	Germany	1964	2.49
6	Germany	1965	2.48
7	Germany	1966	2.44
8	Germany	1967	2.37
9	Germany	1968	2.28
10	Germany	1969	2.17

pivot_longer

```
new_tidy_data <- pivot_longer(data       = wide_data,
                              cols       = `1960`:`2015`,
                              names_to   = "year",
                              values_to = "fertility")
```

- Another way to write this code is to specify the column(s) that will not be included in the pivot.

```
new_tidy_data <- pivot_longer(data       = wide_data,
                              cols       = -country,
                              names_to   = "year",
                              values_to = "fertility")
```

- Note that `pivot_longer` assumes that column names are *characters*.

pivot_longer

```
new_tidy_data <- pivot_longer(data       = wide_data,
                              cols       = `1960`:`2015`,
                              names_to   = "year",
                              values_to = "fertility")
```

- Another way to write this code is to specify the column(s) that will not be included in the pivot.

```
new_tidy_data <- pivot_longer(data       = wide_data,
                              cols       = -country,
                              names_to   = "year",
                              values_to = "fertility")
```

- Note that `pivot_longer` assumes that column names are *characters*.

```
new_tidy_data <- pivot_longer(data       = wide_data,
                              cols       = -country,
                              names_to   = "year",
                              values_to = “fertility") %>%
                 mutate(year = as.integer(year))
```

`pivot_wider`

- It is sometimes useful to convert tidy data into wide data
- We may want to do this in an intermediate step of the wrangling process
- The function `pivot_wider`, also from the ***tidyr*** package, allow us to do just that

```
pivot_wider(data frame, names_from, values_from)
```

- `data frame`: Name of data frame of interest
- `names_from`: variable whose observations will be used as column names
- `values_from`: variable whose observations will be used to fill the cells

pivot_wider

```
new_wide_data <- new_tidy_data %>%
                  pivot_wider(names_from  = year,
                              values_from = fertility)
```

country	1960	1961	1962	1963	1964	1965	1966
Germany	2.41	2.44	2.47	2.49	2.49	2.48	2.44
South Korea	6.16	5.99	5.79	5.57	5.36	5.16	4.99

separate

- Consider the following dataset:

```
path     <- system.file("extdata", package="dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")
raw_dat  <- read_csv(filename)
select(raw_dat, 1:5)
```

	country	1960_fertility	1960_life_expectancy	1961_fertility	1961_life_expectancy
1	Germany	2.41	69.26	2.44	69.85
2	South Korea	6.16	53.02	5.99	53.75

- Note the dataset is in wide format
- It contains two variables: *life expectancy* and *fertility*, where the column names denote the encoding
- This is not recommended, but it is quite common
- Let's fix this

separate

- Let’s start by using the `pivot_longer` function:

```
dat <- raw_dat %>% pivot_longer(-country)
```

	country	name	value
1	Germany	1960_fertility	2.41
2	Germany	1960_life_expectancy	69.26
3	Germany	1961_fertility	2.44
4	Germany	1961_life_expectancy	69.85
5	Germany	1962_fertility	2.47

- This is not quite in tidy format
- Note that each observation is associated with two, not one, rows
- Let’s use the `separate` function to separate the year and variable in the *name* column

`separate`

- Encoding multiple variables in a single column is very common in practice
- The separate function allow us to tackle this problem

```
separate(col, into, sep)
```

- `col`: Name of the column to be separated
- `into`: names for the new columns
- `sep:character that separates the variables`

separate

- Let's start by using the `pivot_longer` function
- Use the `separate` function

```
dat <- raw_dat %>% pivot_longer(-country)

dat %>% separate(col = name, into = c("year", "name"), sep = "_")
```

	country	year	name	value
1	Germany	1960	fertility	2.41
2	Germany	1960	life	69.26
3	Germany	1961	fertility	2.44
4	Germany	1961	life	69.85
5	Germany	1962	fertility	2.47

- Is this right? Any comments?

separate

- The function does separate the values, but *life_expectancy* was truncated to *life*.
- Let's use another argument, *extra,* to take care of this

```
dat <- raw_dat %>% pivot_longer(-country)

dat %>% separate(col = name, into = c("year", "name"), sep = "_", extra = "merge")
```

	country	year	name	value
1	Germany	1960	fertility	2.41
2	Germany	1960	life_expectancy	69.26
3	Germany	1961	fertility	2.44
4	Germany	1961	life_expectancy	69.85
5	Germany	1962	fertility	2.47

- Are we done?

separate

- The function does separate the values, but *life_expectancy* was truncated to *life*.
- Let's use another argument, *extra,* to take care of this

```
dat <- raw_dat %>% pivot_longer(-country)

dat %>% separate(col = name, into = c("year", "name"), sep = "_", extra = "merge")
```

	country	year	name	value
1	Germany	1960	fertility	2.41
2	Germany	1960	life_expectancy	69.26
3	Germany	1961	fertility	2.44
4	Germany	1961	life_expectancy	69.85
5	Germany	1962	fertility	2.47

- Are we done? Not yet
- We need to create a column for each variable. Ideas?

separate

- Let's start by using the `pivot_longer` function
- Use the `separate` function
- Use the `pivot_wider` function

```
dat %>%
   separate(col = name, into = c("year", "name"), sep = "_", extra = "merge") %>%
   pivot_wider()
```

	country	year	fertility	life_expectancy
1	Germany	1960	2.41	69.26
2	Germany	1961	2.44	69.85
3	Germany	1962	2.47	70.01
4	Germany	1963	2.49	70.10
5	Germany	1964	2.49	70.66

separate

- Notice the progress we made in this simple example

Raw data

	country	name	value
1	Germany	1960_fertility	2.41
2	Germany	1960_life_expectancy	69.26
3	Germany	1961_fertility	2.44
4	Germany	1961_life_expectancy	69.85
5	Germany	1962_fertility	2.47

Tidy data

	country	year	fertility	life_expectancy
1	Germany	1960	2.41	69.26
2	Germany	1961	2.44	69.85
3	Germany	1962	2.47	70.01
4	Germany	1963	2.49	70.10
5	Germany	1964	2.49	70.66

References

1. Introduction to Data Science: Data analysis and prediction algorithms with R by Rafael A. Irizarry, Chapter 21. https://rafalab.github.io/dsbook/
2. R for Data Science by Grolemund & Wickham, Chapter 12. https://r4ds.had.co.nz/index.html

Referencias en español:

1. Introducción a la Ciencia de Datos: Análisis de datos y algoritmos de predicción con R por Rafael A. Irizarry, Capítulo 21. https://rafalab.github.io/dslibro/
2. R para Ciencia de Datos por Grolemund & Wickham, Capítulo 12. https://es.r4ds.hadley.nz

Your turn!

[Click here for the class website]