

# APPLICATIONS



OF DATA SCIENCE

# Tidy Data Wrangling - Part B

## Applications of Data Science - Class 3

Giora Simchoni

[gsimchoni@gmail.com](mailto:gsimchoni@gmail.com) and add #dsapps in subject

Stat. and OR Department, TAU

2023-02-23

APPLICATIONS



OF DATA SCIENCE

# Joining Tables

APPLICATIONS



OF DATA SCIENCE

# Detour: The Starwars Dataset(s)

APPLICATIONS



Of DATA SCIENCE

# The Starwars Dataset(s)

planets	
<b>planet_id</b>	int
name	varchar
diameter	double
climate	varchar
gravity	varchar
population	double

characters	
<b>character_id</b>	int
name	varchar
gender	varchar
height	varchar
eye_color	varchar
film_id	int
homeworld_id	int

films	
<b>film_id</b>	int
<b>episode_id</b>	int
title	varchar
release_date	date
director	varchar
producer	varchar
opening_crawl	varchar

```
sw_tables <- read_rds("../data/sw_tables.rds")
characters <- sw_tables$characters
planets <- sw_tables$planets
films <- sw_tables$films
```



What are the advantages/disadvantages of storing data in such a way?

```
glimpse(characters)
```

```
## Rows: 173
## Columns: 14
## $ character_id <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
## $ name <chr> "Luke Skywalker", "Luke Skywalker", "Luke Skywalker"
## $ gender <chr> "male", "male", "male", "male", NA, NA, NA,
## $ homeworld_id <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 8, 8, 8, 8, 8, 8,
## $ height <dbl> 172, 172, 172, 172, 172, 167, 167, 167, 167, 167, 167, 167, 167, 167, 167, 167, 167
## $ mass <dbl> 77, 77, 77, 77, 77, 75, 75, 75, 75, 75, 75, 75, 75, 32, 32,
## $ hair_color <chr> "blond", "blond", "blond", "blond", "blond", NA, NA
## $ skin_color <chr> "fair", "fair", "fair", "fair", "fair", "gold", "go
## $ eye_color <chr> "blue", "blue", "blue", "blue", "blue", "yellow", "
## $ birth_year <dbl> 19.0, 19.0, 19.0, 19.0, 19.0, 112.0, 112.0, 112.0,
## $ film_id <dbl> 1, 2, 3, 6, 7, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6,
## $ species <list> "http://swapi.co/api/species/1/", "http://swapi.co
## $ vehicles <list> <"http://swapi.co/api/vehicles/14/", "http://swapi
## $ starships <list> <"http://swapi.co/api/starships/12/", "http://swap
```

## glimpse(planets)

glimpse(films)

```
## Rows: 7
## Columns: 7
## $ film_id      <dbl> 1, 2, 3, 4, 5, 6, 7
## $ title        <chr> "A New Hope", "The Empire Strikes Back", "Return o
## $ episode_id    <int> 4, 5, 6, 1, 2, 3, 7
## $ opening_crawl <chr> "It is a period of civil war.\r\nRebel spaceships,
## $ director      <chr> "George Lucas", "Irvin Kershner", "Richard Marquand"
## $ producer       <chr> "Gary Kurtz, Rick McCallum", "Gary Kutz, Rick McCal
## $ release_date   <chr> "1977-05-25", "1980-05-17", "1983-05-25", "1999-05-
```



# End of Detour

APPLICATIONS



OF DATA SCIENCE

## Q: Which characters appear in SW films directed by George Lucas?

```
unique(
  characters$name[
    characters$film_id %in%
      films$film_id[films$director == "George Lucas"]
  ]
)

# or dplyr approach:
characters %>%
  filter(film_id %in% (films %>%
    filter(director=="George Lucas") %>%
    pull(film_id))) %>%
  pull(name) %>%
  unique()
```

First problem with this approach: code gets messier and messier, prone to bugs and hard to debug.

## Q: Which characters, whose homeworld is Alderaan, appear in SW films directed by George Lucas?

```
unique(  
  characters$name[  
    characters$film_id %in%  
      films$film_id[films$director == "George Lucas"] &  
    characters$homeworld_id ==  
      planets$planet_id[planets$name == "Alderaan"]  
  ]  
)  
  
# or dplyr approach:  
characters %>%  
  filter(film_id %in% (films %>%  
    filter(director=="George Lucas") %>%  
    pull(film_id)),  
  homeworld_id == (planets %>%  
    filter(name == "Alderaan") %>%  
    pull(planet_id))) %>%  
  pull(name) %>%  
  unique()  
  
# [1] "Leia Organa"          "Bail Prestor Organa" "Raymus Antilles"
```

## Now imagine these two tables

lines: each film, each scene, each minute, each character, the line

```
lines <- tibble::tribble(
  ~film_id, ~scene_id, ~minute_id, ~character_id,
  1, 1, 1, 23,
  1, 1, 2, 15, "somet
  1, 1, 2, 23, "something somet
  1, 1, 3, 15,
  1, 1, 3, 23,
  1, 1, 4, 8, "wha
)
```

locations: for each film, each scene, each minute, its location

```
locations <- tibble::tribble(
  ~film_id, ~scene_id, ~minute_id, ~location,
  1, 1, 1, "spaceship",
  1, 1, 2, "Alderaan, outdoors",
  1, 1, 3, "spaceship",
  1, 1, 4, "bar"
)
```

## Q: Which characters say "something" in a bar?

- filter only lines which contain "something"
- filter only "bar" locations and then...
- if the two filtered tables match on film, scene and minute...
- we take the unique characters
- but how do we match? `for` loop\*

→ Clearly, second problem with this approach: it doesn't generalize well to more complex scenarios, where we need to match on multiple criteria

And third problem: speed (but only in a complex scenario!)

\* There *is* a way without using a for loop, without joining, still not recommended.

# inner\_join()

(Inner) Joining two tables:

```
characters %>%
  inner_join(films) %>%
  select(character_id, name, film_id, title, director) %>%
  head(7)

## Joining with `by = join_by(film_id)`

## # A tibble: 7 × 5
##   character_id name      film_id title                director
##       <int> <chr>     <dbl> <chr>                <chr>
## 1           1 Luke Skywalker     1 A New Hope        George Lucas
## 2           1 Luke Skywalker     2 The Empire Strikes Back Irvin Kershner
## 3           1 Luke Skywalker     3 Return of the Jedi Richard Marquand
## 4           1 Luke Skywalker     6 Revenge of the Sith George Lucas
## 5           1 Luke Skywalker     7 The Force Awakens J. J. Abrams
## 6           2 C-3PO             1 A New Hope        George Lucas
## 7           2 C-3PO             2 The Empire Strikes Back Irvin Kershner
```

## (Inner) Joining multiple tables:

```
characters %>%
  inner_join(films) %>%
  inner_join(planets, by = c("homeworld_id" = "planet_id")) %>%
  select(character_id, name.x, film_id, title, director, name.y, c

## Joining with `by = join_by(film_id)` 

## # A tibble: 7 × 7
##   character_id name.x      film_id title          director name.y
##       <int> <chr>      <dbl> <chr>          <chr>    <chr>
## 1           1 Luke Skywalker     1 A New Hope George ... Tatoo...
## 2           1 Luke Skywalker     2 The Empire Strike... Irvin K... Tatoo...
## 3           1 Luke Skywalker     3 Return of the Jedi Richard... Tatoo...
## 4           1 Luke Skywalker     6 Revenge of the Si... George ... Tatoo...
## 5           1 Luke Skywalker     7 The Force Awakens J. J. A... Tatoo...
## 6           2 C-3PO              1 A New Hope George ... Tatoo...
## 7           2 C-3PO              2 The Empire Strike... Irvin K... Tatoo...
```

## (Inner) Joining on multiple keys:

```
lines %>%
  inner_join(locations, by = c("film_id", "scene_id", "minute_id"))
```

```
## # A tibble: 6 × 6
##   film_id scene_id minute_id character_id line
##       <dbl>     <dbl>      <dbl>        <dbl> <chr>
## 1         1         1          1            1  blah blah blah
## 2         1         1          1            2  something something
## 3         1         1          1            2  something something to you to
## 4         1         1          1            3        <NA>
## 5         1         1          1            3        <NA>
## 6         1         1          1            4  whatever whatever
```



The base R function for joining is `merge()` which tends to be slower.

# What would `inner_join()` do?

Q: Which characters appear in SW films directed by George Lucas?

```
# naive
characters %>%
  inner_join(films, by = "film_id") %>%
  filter(director == "George Lucas") %>%
  pull(name) %>%
  unique()

# smarter
characters %>%
  inner_join(films %>% filter(director == "George Lucas"),
             by = "film_id") %>%
  pull(name) %>%
  unique()
```



What else could you do to make `inner_join`'s life easier?

# What would `inner_join()` do?

Q: Which characters, whose homeworld is Alderaan, appear in SW films directed by George Lucas?

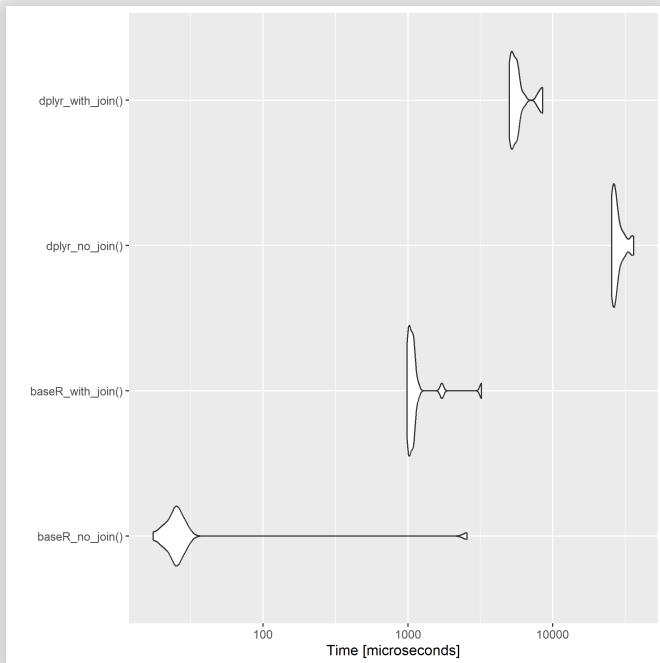
```
characters %>%
  inner_join(films %>% filter(director == "George Lucas"),
             by = "film_id") %>%
  inner_join(planets %>% filter(name == "Alderaan"),
             by = c("homeworld_id" = "planet_id"),
             suffix = c("_char", "_planet")) %>%
  pull(name_char) %>%
  unique()
```

```
## [1] "Leia Organa"          "Bail Prestor Organa" "Raymus Antilles"
```

# Note of caution: Join is not always faster!

```
baseR_no_join <- function() {  
  unique(characters$name[characters$film_id %in% films$film_id[fi]  
}  
  
baseR_with_join <- function() {  
  unique(merge(characters, films[films$director == "George Lucas",  
}  
  
dplyr_no_join <- function() {  
  characters %>%  
    filter(film_id %in% (films %>%  
      filter(director=="George Lucas") %>%  
      pull(film_id))) %>%  
    pull(name) %>%  
    unique()  
}  
  
dplyr_with_join <- function() {  
  characters %>%  
    inner_join(films %>% filter(director=="George Lucas"), by = "f  
    pull(name) %>%  
    unique()  
}
```

```
library(microbenchmark)
res <- microbenchmark(baseR_no_join(), baseR_with_join(),
                      dplyr_no_join(), dplyr_with_join(), times =
autoplot(res)
```



# So when does it shine?

- See [this](#) StackOverflow question and answer for a good example\*
- But in general: when data gets bigger, when multiple keys are involved

\* Yes, I know, I answered my own question, 🤦

## Other types of Joins?

Definitely, read [here](#), [here](#) and [here](#) about:

- `left_join()`
- `right_join()`
- `full_join()`
- `anti_join()`
- `semi_join()`

# All Joins

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



A	B	D	
b	u	3	
c	v	2	
d	w	1	



A	B	C	D
b	u	2	3
c	v	3	2

inner\_join()

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



A	B	D	
b	u	3	
c	v	2	
d	w	1	



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

left\_join()

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



A	B	D	
b	u	3	
c	v	2	
d	w	1	



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

right\_join()

# All Joins

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

A	B	D	
b	u	3	
c	v	2	
d	w	1	

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

full\_join()

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

A	B	D	
b	u	3	
c	v	2	
d	w	1	

A	B	C	
b	u	2	
c	v	3	

semi\_join()

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

A	B	D	
b	u	3	
c	v	2	
d	w	1	

A	B	C	
a	t	1	

anti\_join()

# Tidying Tables

APPLICATIONS



OF DATA SCIENCE

# Detour: The Migration Dataset

APPLICATIONS



OF DATA SCIENCE

# The Migration Dataset

- Straight from the [UN, Dept. of Economic and Social Affairs, Population Division](#)
- "Monitoring global population trends"
- For each country, how many (men, women and total) migrated to each country
- In 1990, 1995, 2000, 2005, 2010, 2015, 2019
- How would *you* give access to these data?

# You want dirty data? Well dirty data costs!

A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	2	3	4	5	6	7	8	9	10	11	12	13	14
													
		United Nations Population Division Department of Economic and Social Affairs											
		Workbook: UN_MigrantStockByOriginAndDestination_2019.xlsx											
		<b>Table 1. Total migrant stock at mid-year by origin and by major area, region, country or area of destination, 1990-2019</b>											
		Copyright © 2019 by United Nations, made available under a Creative Commons license CC BY 3.0 IGO: <a href="http://creativecommons.org/licenses/by/3.0/igo/">http://creativecommons.org/licenses/by/3.0/igo/</a>											
		Suggested citation: United Nations, Department of Economic and Social Affairs, Population Division (2019). International Migrant Stock: 2019 (United Nations database, <i>PORDIMIGStockRev.2019</i> ).											
Year	Sort order	Major area, region, country or area of destination	Notes	Code	Type of data (a)	Total	Other South	Other North	Afghanistan	Albania	Algeria	American Samoa	Andorra
1990	1990001	<b>WORLD</b>	900			<b>153,011,473</b>	<b>6,548,526</b>	<b>2,366,800</b>	<b>6,823,350</b>	<b>180,284</b>	<b>921,727</b>	<b>2,041</b>	<b>3,792</b>
1990	1990002	<b>UN development groups</b>				..	..	..	..	..	..	..	..
1990	1990003	More developed regions	b	901		<b>82,767,216</b>	<b>3,385,103</b>	<b>1,077,179</b>	<b>119,386</b>	<b>177,986</b>	<b>867,015</b>	<b>1,027</b>	<b>3,737</b>
1990	1990004	Less developed regions	c	902		<b>70,244,257</b>	<b>3,163,423</b>	<b>1,289,621</b>	<b>6,703,964</b>	<b>2,298</b>	<b>54,712</b>	<b>1,014</b>	<b>55</b>
1990	1990005	Least developed countries	d	941		11,060,221	482,753	239,756	0	0	5,522	0	0
1990	1990006	Less developed regions, excluding least developed countries	e	934		53,164,036	2,880,670	1,049,865	6,703,964	2,298	49,090	1,014	55
1990	1990007	<b>World Bank income groups</b>				..	..	..	..	..	..	..	..
1990	1990008	High-income countries	e	1502		<b>77,802,968</b>	<b>3,803,597</b>	<b>1,223,239</b>	<b>268,933</b>	<b>131,449</b>	<b>886,598</b>	<b>68</b>	<b>3,748</b>
1990	1990009	Middle-income countries	e	1517		<b>65,124,386</b>	<b>1,877,678</b>	<b>987,475</b>	<b>6,544,932</b>	<b>48,636</b>	<b>30,513</b>	<b>1,973</b>	<b>44</b>
1990	1990010	Upper-middle-income countries	e	1502		33,265,549	957,059	376,006	3,152,557	48,298	4,719	1,954	42
1990	1990011	Lower-middle-income countries	e	1501		31,838,817	920,619	611,459	3,331,975	538	25,794	19	2
1990	1990012	Low-income countries	e	1500		<b>9,790,238</b>	<b>850,549</b>	<b>145,493</b>	<b>8,485</b>	<b>0</b>	<b>4,370</b>	<b>0</b>	<b>0</b>
1990	1990013	No income group available	e			294,001	16,702	10,593	0	0	255	0	0
1990	1990014	<b>Geographic regions</b>				..	..	..	..	..	..	..	..
1990	1990015	Africa	903			<b>15,689,666</b>	<b>807,899</b>	<b>258,638</b>	<b>1,037</b>	<b>504</b>	<b>31,807</b>	<b>0</b>	<b>0</b>
1990	1990016	Asia	935			<b>48,209,949</b>	<b>2,005,398</b>	<b>961,154</b>	<b>6,702,728</b>	<b>1,291</b>	<b>20,924</b>	<b>0</b>	<b>0</b>
1990	1990017	Europe	908			<b>49,608,231</b>	<b>432,964</b>	<b>871,477</b>	<b>82,736</b>	<b>170,977</b>	<b>857,978</b>	<b>963</b>	<b>3,404</b>
1990	1990018	Latin America and the Caribbean	904			7,161,371	344,362	64,989	199	503	863	0	55
1990	1990019	Northern America	905			<b>27,610,408</b>	<b>2,923,653</b>	<b>198,963</b>	<b>33,765</b>	<b>6,025</b>	<b>8,372</b>	<b>0</b>	<b>0</b>
1990	1990020	Oceania	909			<b>4,731,848</b>	<b>34,250</b>	<b>11,579</b>	<b>2,885</b>	<b>984</b>	<b>1,783</b>	<b>1,078</b>	<b>333</b>
1990	1990021	<b>Sustainable Development Goal (SDG) regions</b>				..	..	..	..	..	..	..	..

CONTENTS | **Table 1** | Table 2 | Table 3 | ANNEX | NOTES | +

# What's untidy about the migration Excel file?

- It's an Excel file 😠
- Multiple sheets
- Color coded, font coded (bold), space coded!
- Logo, free text in header lines
- Merged cells
- French letters (anything but [A-Za-z] can break code)
- Spaces, parentheses in column names
- Different NA values: "..", "-"
- Variable "country\_dest" contains sub-total and totals for categories, continents...
- Variable "country\_orig" violates Tidy rule no. 1: not in its own column

# "Today Only, The Landlord Went Nuts!"

- ~~It's an Excel file~~ 😠
- ~~Multiple sheets~~
- ~~Color coded, font coded (bold), space coded!~~
- ~~Logo, free text in header lines~~
- ~~Merged cells~~
- ~~French letters (anything but [A-Z a-z] can break code)~~
- ~~Spaces, parentheses in column names~~
- ~~Different NA values: "", ""~~
- ~~Variable "country\_dest" contains sub-total and totals for categories, continents...~~
- Variable "country\_orig" violates Tidy rule no. 1: not in its own column

# The much nicer migration table

```
migration <- read_rds("../data/migration.rds")  
  
migration  
  
## # A tibble: 3,248 × 236  
##   gender year  code country_dest afghanistan albania algeria american_...  
##   <chr>   <dbl> <dbl> <chr>          <dbl>    <dbl>    <dbl>  
## 1 men     1990   108 burundi          0        0        0  
## 2 men     1990   174 comoros          0        0        0  
## 3 men     1990   262 djibouti         0        0        0  
## 4 men     1990   232 eritrea          0        0        0  
## 5 men     1990   231 ethiopia         0        0        0  
## 6 men     1990   404 kenya            0        0        0  
## 7 men     1990   450 madagascar       0        0        0  
## 8 men     1990   454 malawi           0        0        0  
## 9 men     1990   480 mauritius        0        0        0  
## 10 men    1990   175 mayotte          0        0        0  
## # ... with 3,238 more rows, and 228 more variables: andorra <dbl>, angola ...  
## # anguilla <dbl>, antigua_and_barbuda <dbl>, argentina <dbl>, armenia ...  
## # aruba <dbl>, australia <dbl>, austria <dbl>, azerbaijan <dbl>, ...  
## # bahamas <dbl>, bahrain <dbl>, bangladesh <dbl>, barbados <dbl>, ...  
## # belarus <dbl>, belgium <dbl>, belize <dbl>, benin <dbl>, bermuda <db...  
## # bhutan <dbl>, bolivia_plurinational_state_of <dbl>, ...  
## # ... with 228 more variables:
```

# It's not right, but it's Ok:

- How many men immigrated from Russia to Israel in 1990?

```
migration %>%
  filter(country_dest == "israel", year == 1990, gender == "men")
  pull(russian_federation)
```

```
## [1] 80450
```

- How many men immigrated from Israel to Russia in 1990?

```
migration %>%
  filter(country_dest == "russian_federation", year == 1990, gender == "men")
  pull(israel)
```

```
## [1] 1395
```

# It would have been much nicer to have:

```
migration %>%
  filter(country_orig == "israel", country_dest == "russian_fedrat"
        year == 1990, gender == "men") %>%
  pull(n_migrants)
```

Then we could put it in a (simpler) function and call the opposite:

```
get_1way_migration <- function(orig, dest, gen, .year) {
  migration %>%
    filter(country_orig == orig, country_dest == dest,
          year == .year, gender == gen) %>%
    pull(n_migrants)
}

get_1way_migration("russian_federation", "israel")
```

# End of Detour

APPLICATIONS



OF DATA SCIENCE

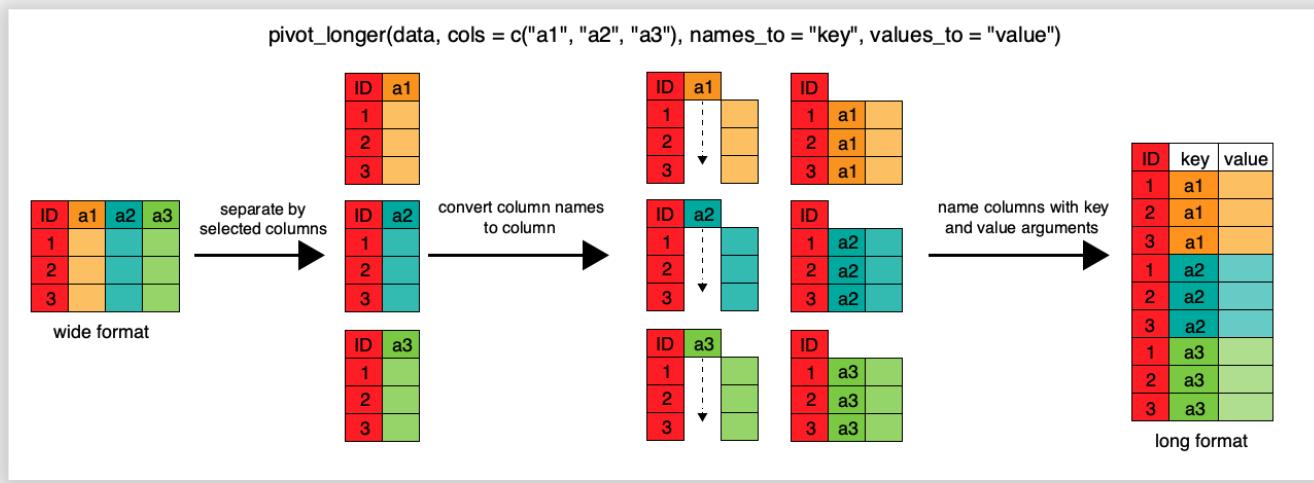
# pivot\_longer()

```
migration_long <- migration %>%
  pivot_longer(cols = -c(1:4),
               names_to = "country_orig",
               values_to = "n_migrants")
```

```
migration_long
```

```
## # A tibble: 753,536 × 6
##   gender year code country_dest country_orig      n_migrants
##   <chr>   <dbl> <dbl> <chr>          <chr>            <dbl>
## 1 men     1990    108 burundi      afghanistan        0
## 2 men     1990    108 burundi      albania           0
## 3 men     1990    108 burundi      algeria           0
## 4 men     1990    108 burundi      american_samoa    0
## 5 men     1990    108 burundi      andorra          0
## 6 men     1990    108 burundi      angola            0
## 7 men     1990    108 burundi      anguilla          0
## 8 men     1990    108 burundi      antigua_and_barbuda 0
## 9 men     1990    108 burundi      argentina         0
## 10 men    1990    108 burundi     armenia           0
## # ... with 753,526 more rows
```

# What sorcery is this?



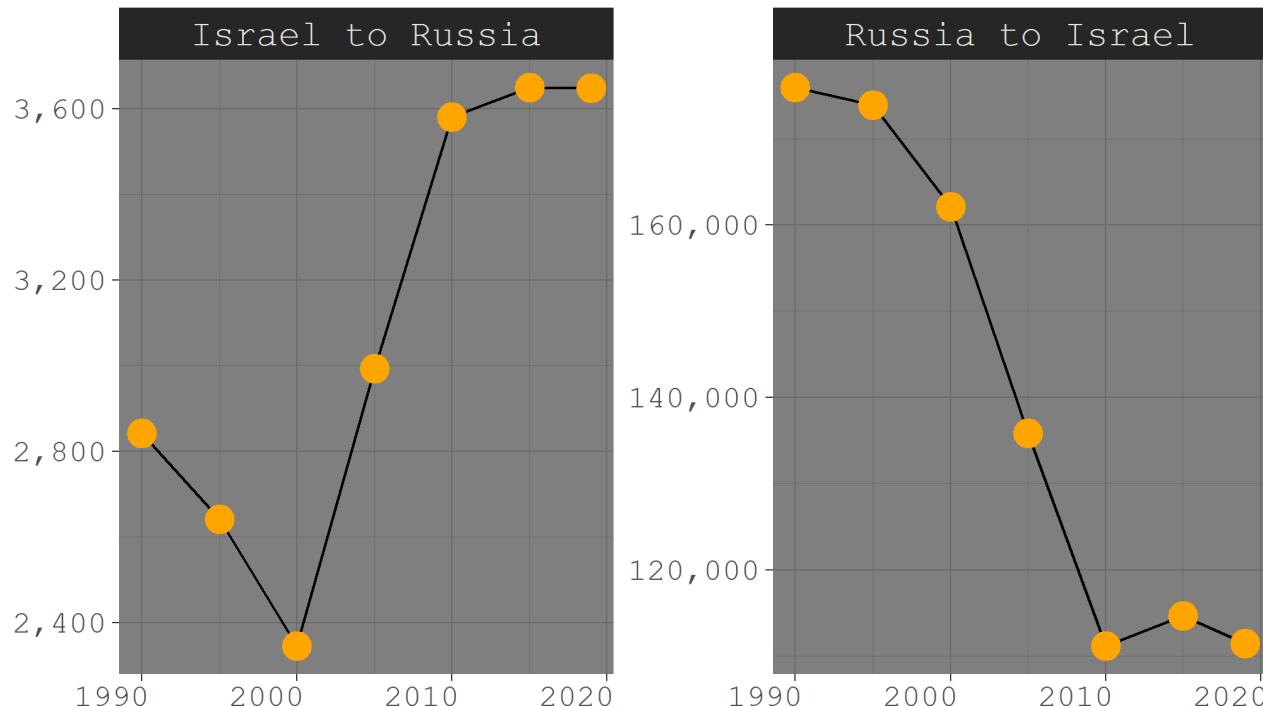
Source: [The Carpentries](#)

## pivot\_wider()

# Where is this going to?

```
get_1way_migration <- function(.country_dest, .country_orig) {  
  migration_long %>%  
    filter(country_dest == .country_dest, country_orig == .country_orig)  
    group_by(year) %>%  
    tally(n_migrants) %>%  
    mutate(direction = str_c(.country_orig, " to ", .country_dest))  
}  
get_2way_migration <- function(country_a, country_b) {  
  a2b <- get_1way_migration(country_b, country_a)  
  b2a <- get_1way_migration(country_a, country_b)  
  bind_rows(a2b, b2a)  
}  
  
get_2way_migration("israel", "russian_federation") %>%  
  ggplot(aes(year, n)) + geom_line() +  
  geom_point(color = "orange", size = 5) +  
  labs(x = "", y = "", title = "Israel-Russia Yearly No. of immigrants") +  
  theme_dark() +  
  facet_wrap(~direction, scales = "free", labeller = labeller(direction =  
    scale_y_continuous(labels = scales::comma_format()) +  
    theme(axis.text = element_text(size = 12, hjust=0.9, family = "mono"),  
          strip.text.x = element_text(size = 14, family = "mono"),  
          plot.title = element_text(hjust = 0.5, size = 18, family = "mono")))
```

## Israel-Russia Yearly No. of immigrants



# Some more useful Tidying up verbs

Remember?

```
table3 <- read_rds("../data/tidy_tables.rds")$table3  
  
table3  
  
## # A tibble: 315 × 3  
##   religion      yob pct_straight  
##   <chr>        <dbl> <chr>  
## 1 atheist      1950  26/29  
## 2 buddhist     1950  6/6  
## 3 christian    1950  28/32  
## 4 hindu        1950  0/0  
## 5 jewish       1950  21/24  
## 6 muslim        1950  0/0  
## 7 unspecified  1950  71/76  
## 8 atheist      1951  31/33  
## 9 buddhist     1951  11/11  
## 10 christian   1951  23/24  
## # ... with 305 more rows
```

# **separate\_wider\_delim()**

(Superseded `separate()`)

```
table3_tidy <- table3 %>%
  separate_wider_delim(pct_straight, delim = "/",
                        names = c("straight", "total"))

table3_tidy
```

```
## # A tibble: 315 × 4
##   religion      yob straight total
##   <chr>        <dbl>  <chr>    <chr>
## 1 atheist       1950   26     29
## 2 buddhist      1950   6      6
## 3 christian     1950   28     32
## 4 hindu         1950   0      0
## 5 jewish        1950   21     24
## 6 muslim         1950   0      0
## 7 unspecified   1950   71     76
## 8 atheist       1951   31     33
## 9 buddhist      1951   11     11
## 10 christian    1951   23     24
## # ... with 305 more rows
```

# unite()

(Though see a much more generalizable approach with purrrr)

```
table3_tidy %>%  
  unite(col = "pct_straight", straight, total, sep = "/")
```

```
## # A tibble: 315 × 3  
##   religion      yob pct_straight  
##   <chr>        <dbl> <chr>  
## 1 atheist      1950 26/29  
## 2 buddhist     1950 6/6  
## 3 christian    1950 28/32  
## 4 hindu        1950 0/0  
## 5 jewish       1950 21/24  
## 6 muslim        1950 0/0  
## 7 unspecified  1950 71/76  
## 8 atheist      1951 31/33  
## 9 buddhist     1951 11/11  
## 10 christian   1951 23/24  
## # ... with 305 more rows
```

# Iteration without looping

APPLICATIONS



OF DATA SCIENCE

# Fun fact: you're already not-looping!

- Say you want the lengths of all of these strings:

```
strings_vec <- c("I'm feeling fine", "I'm perfectly OK",
                 "Nothing is wrong!")
```

- Do you `for` loop?

```
strings_len <- numeric(length(strings_vec))
for (i in seq_along(strings_vec)) {
  strings_len[i] <- nchar(strings_vec[i])
}
strings_len
```

```
## [1] 16 16 17
```

- No, you use R's vectorized functions nature:

```
nchar(strings_vec)
```

```
## [1] 16 16 17
```

- In case you were wondering:

```
microbenchmark(nchar_loop(), nchar_vectorized())
```

```
## Unit: nanoseconds
##           expr   min    lq      mean    median     uq      max neval
##   nchar_loop() 2200 2301 36360.07  2301 2401 3394201    100
## nchar_vectorized()   600   701 13650.95    701   801 1284502    100
```

- So why should iterating through a `data.frame` columns or rows be any different?

# The problem:

```
okcupid <- read_csv("~/okcupid.csv.zip", col_types = cols())
okcupid %>% select(essay0:essay2) %>% head(3)
```

```
## # A tibble: 3 × 3
##   essay0                           essay
##   <chr>                            <chr>
## 1 "about me:<br />\n<br />\ni would love to think that i was some..." "curr.
## 2 "i am a chef: this is what that means.<br />\n1. i am a workaho..." "dedi.
## 3 "i'm not ashamed of much, but writing public text on an online\..." "i ma.
```

- I want to apply some transformation to one or more columns (say the length of each `essay` question for each user)
- I want to apply some transformation to one or more rows (say the average length of all `essay` questions for each user)
- BTW, you can always use a `for` loop, see how it is done [here](#)

# The `apply` family

- There *is* a solution in base R
  - `apply()`
  - `sapply()`
  - `lapply()`
  - `tapply()`
  - `vapply()`
  - `mapply()`
- What do you think is the issue with these? See [this](#).

# I don't for, I purrr

The `purrr` package provides a set of functions to make iteration easier:

- No boilerplate code for looping --> less looping bugs
- Focus on the function, the action, not the plumbing
- Generally faster (implemented in C)
- Definitely more clear, concise and elegant code

TBH, I'm addicted 😊

# You get a `map()`, you get a `map()`!

Single	Two	Multiple	Returns	Of
<code>map()</code>	<code>map2()</code>	<code>pmap()</code>	<code>list</code>	?
<code>map_lgl()</code>	<code>map2_lgl()</code>	<code>pmap_lgl()</code>	<code>vector</code>	<code>logical</code>
<code>map_chr()</code>	<code>map2_chr()</code>	<code>pmap_chr()</code>	<code>vector</code>	<code>character</code>
<code>map_int()</code>	<code>map2_int()</code>	<code>pmap_int()</code>	<code>vector</code>	<code>integer</code>
<code>map_dbl()</code>	<code>map2_dbl()</code>	<code>pmap_dbl()</code>	<code>vector</code>	<code>double</code>
<code>map_dfr()</code>	<code>map2_dfr()</code>	<code>pmap_dfr()</code>	<code>tibble</code>	?

Where "Single" means "single vector/column input", "Two" means "two vectors/columns input" etc.

(Tip of the iceberg really, I want you to survive this slide)

# Example1: Vectorizing a Function

Take a clearly not-vectorized function:

```
my_func <- function(x) {  
  if (x %% 2 == 0) return("even")  
  "odd"  
}  
  
my_func(10)  
  
## [1] "even"  
  
my_func(1:5)  
  
## Error in if (x%%2 == 0) return("even"): the condition has length > 1
```



This is a silly example, do you know how to easily vectorize this function?

`map()` will always return a list:

```
map(1:3, my_func)
```

```
## [[1]]  
## [1] "odd"  
##  
## [[2]]  
## [1] "even"  
##  
## [[3]]  
## [1] "odd"
```

```
1:3 %>% map(my_func)
```

```
## [[1]]  
## [1] "odd"  
##  
## [[2]]  
## [1] "even"  
##  
## [[3]]  
## [1] "odd"
```

`map_chr()` will always return a vector of character:

```
map_chr(1:3, my_func)  
## [1] "odd"  "even" "odd"
```

```
1:3 %>% map_chr(my_func)  
## [1] "odd"  "even" "odd"
```

But here is the beautiful thing:

```
my_func_vectorized <- function(vec) map_chr(vec, my_func)  
my_func_vectorized(1:3)  
## [1] "odd"  "even" "odd"
```

Look Ma, no loops!

## Example2: Complex `mutate()`

Manager: Add me a column, for each OkCupid user, whether he/she's above average height.

```
is_above_average_height <- function(sex, height_cm) {  
  if (sex == "m") height_cm > 180 else height_cm > 165  
}  
  
okcupid <- okcupid %>%  
  mutate(is_tall = map2_lgl(sex, height_cm, is_above_average_height))  
  
okcupid %>% select(sex, height_cm, is_tall) %>%  
  group_by(sex) %>% slice_sample(n = 3)  
  
## # A tibble: 6 × 3  
## # Groups:   sex [2]  
##   sex     height_cm is_tall  
##   <chr>    <dbl> <lgl>  
## 1 f        160. FALSE  
## 2 f        168. TRUE  
## 3 f        173. TRUE  
## 4 m        165. FALSE  
## 5 m        178. FALSE  
## 6 m        173. FALSE
```

## Example2: You could even supply args

```
is_above_average_height <- function(sex, height_cm, men_avg, women_avg)
  if(sex == "m") height_cm > men_avg else height_cm > women_avg
}

okcupid <- okcupid %>%
  mutate(is_tall = map2_lgl(sex, height_cm, is_above_average_height,
                           men_avg = 180, women_avg = 165))

okcupid %>% select(sex, height_cm, is_tall) %>% group_by(sex) %>%

## # A tibble: 6 × 3
## # Groups:   sex [2]
##   sex    height_cm is_tall
##   <chr>     <dbl> <lgl>
## 1 f          160. FALSE
## 2 f          155. FALSE
## 3 f          165. TRUE
## 4 m          178. FALSE
## 5 m          180. TRUE
## 6 m          173. FALSE
```

## Example2: Anonymous Functions

```
okcupid <- okcupid %>%
  mutate(is_tall = map2_lgl(sex, height_cm,
                            function(x, y) if(x == "m") y > 180 else
```

Heck, you can even:

```
okcupid <- okcupid %>%
  mutate(is_tall = map2_lgl(sex, height_cm,
                            ~if(.x == "m") .y > 180 else .y > 165)
```



There's a thin line between elegance and unreadable undetectable bragging.

# Example 3: Remember our problem?

For each `essay` column, add a column with its length:

```
okcupid %>%
  mutate(across(essay0:essay9, list("len" = str_length))) %>%
  select(starts_with("essay")) %>%
  head(3)

## # A tibble: 3 × 20
##   essay0          essay1 essay2 essay3 essay4 essay5 essay6 essay7 essay
##   <chr>           <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 "about me:<br ... "curr... "maki... "the ... "book... "food... duali... "tryi...
## 2 "i am a chef: ... "dedi... "bein... <NA>   "i am... "deli... <NA>   "i am.
## 3 "i'm not asham... "i ma... "impr... "my l... "okay... "move... <NA>   "view... "when.
## # ... with 10 more variables: essay0_len <int>, essay1_len <int>,
## #   essay2_len <int>, essay3_len <int>, essay4_len <int>, essay5_len <int>
## #   essay6_len <int>, essay7_len <int>, essay8_len <int>, essay9_len <int>
```

`purrr` not needed, like I said, you've already been non-looping!

# Example 3: Input multiple columns

~~For each user, compute the average essay length.~~

Wait, before that, for each user compute the average of the first 3 essays:

```
mean_length_3essay <- function(x, y, z) {  
  mean(str_length(c(x, y, z)), na.rm = TRUE)  
}  
  
okcupid %>%  
  mutate(essay3_avglen = pmap_dbl(  
    list(essay0, essay1, essay2),  
    mean_length_3essay)) %>%  
  select(essay0:essay2, essay3_avglen) %>%  
  head(2)  
  
## # A tibble: 2 × 4  
##   essay0                                essay1 essay2 essay3_avglen  
##   <chr>                                 <chr>  <chr>  <dbl>  
## 1 "about me:<br />\n<br />\ni would love to think t... "curr... "maki...  
## 2 "i am a chef: this is what that means.<br />\n1. ... "dedi... "bein..."
```

OK now, for each user, compute the average essay length:

```
mean_length_essay <- function(...) {  
  mean(str_length(c(...)), na.rm = TRUE)  
}  
  
okcupid %>%  
  mutate(essay_avglen = pmap_dbl(  
    across(starts_with("essay")),  
    mean_length_essay)) %>%  
  select(essay0:essay2, essay_avglen) %>%  
  head(3)
```

```
## # A tibble: 3 × 4  
##   essay0                           essay1 essay2 essa:  
##   <chr>                             <chr>  <chr>  
## 1 "about me:<br />\n<br />\ni would love to think th... "curr... "maki...  
## 2 "i am a chef: this is what that means.<br />\n1. i... "dedi... "bein...  
## 3 "i'm not ashamed of much, but writing public text ... "i ma... "impr...
```

# Example4: Output multiple columns

```
essay0_features <- function(essay0) {  
  contains_love <- str_detect(essay0, "love")  
  contains_obama <- str_detect(essay0, "obama")  
  contains_rel <- str_detect(essay0, "relationship")  
  list(  
    essay0_love = contains_love,  
    essay0_obama = contains_obama,  
    essay0_rel = contains_rel  
  )  
}  
  
okcupid %>% select(essay0) %>% map_dfc(essay0_features) %>% head(4)
```

```
## # A tibble: 4 × 3  
##   essay0_love essay0_obama essay0_rel  
##   <lgl>       <lgl>       <lgl>  
## 1 TRUE        FALSE        FALSE  
## 2 TRUE        FALSE        FALSE  
## 3 TRUE        FALSE        FALSE  
## 4 FALSE       FALSE        FALSE
```

```
# or: map_dfc(okcupid$essay0, essay0_features)
```

Or if you want those features as additional columns, you could bind them with `bind_cols()`:

```
okcupid %>%
  bind_cols(
    okcupid %>% select(essay0) %>% map_dfc(essay0_features)
  ) %>%
  select(age, sex, starts_with("essay0"))

## # A tibble: 59,946 × 6
##       age   sex essay0          essay0_love essay0_obama es...
##   <dbl> <chr> <chr>           <lgl>        <lgl>      <lo...
## 1     22   m "about me:<br />\n<br />\ni ... TRUE        FALSE      FA...
## 2     35   m "i am a chef: this is what t... TRUE        FALSE      FA...
## 3     38   m "i'm not ashamed of much, bu... TRUE        FALSE      FA...
## 4     23   m "i work in a library and go ... FALSE       FALSE      FA...
## 5     29   m "hey how's it going? current... FALSE       FALSE      FA...
## 6     29   m "i'm an australian living in... TRUE        FALSE      FA...
## 7     32   f "life is about the little th... TRUE        FALSE      FA...
## 8     31   f <NA>                         NA          NA        NA
## 9     24   f <NA>                         NA          NA        NA
## 10    37   m "my names jake.<br />\ni'm a... TRUE        FALSE      FA...
## # ... with 59,936 more rows
```

```
# or: okcupid %>% mutate(as_tibble(essay0_features(essay0)))
```

# **rowwise()** and **c\_across()**

The Tidyverse dev team recently acknowledged that the purrr lingo might be a bit much for many users. Enter `rowwise()`:

Before:

```
is_above_average_height <- function(sex, height_cm) {  
  if (sex == "m") height_cm > 180 else height_cm > 165  
}  
  
okcupid %>%  
  mutate(is_tall = map2_lgl(sex, height_cm, is_above_average_height))
```

After:

```
okcupid %>%  
  rowwise() %>%  
  mutate(is_tall = is_above_average_height(sex, height_cm))
```

# **rowwise() and c\_across()**

Before:

```
mean_length_essay <- function(...) {  
  mean(str_length(c(...)), na.rm = TRUE)  
}  
  
okcupid %>%  
  mutate(essay_avglen = pmap_dbl(  
    across(starts_with("essay")),  
    mean_length_essay))
```

After:

```
okcupid %>%  
  rowwise() %>%  
  mutate(essay_avglen =  
    mean_length_essay(c_across(starts_with("essay"))))
```

# Dealing with failure

```
my_func("a")
```

```
## Error in x%%2: non-numeric argument to binary operator
```

Silly example, but:

- (a) When dealing with big data expect the unexpected (input)
- (b) You don't want your app to crash

Look at: `safely()`, `quietly()` and `possibly()` which wrap your code nicely and protect you from the unexpected (crash).

# My favorite: possibly()

```
my_func_safe <- possibly(my_func, otherwise = NA)  
my_func_safe("a")
```

```
## [1] NA
```

```
map_chr(list(1, 2, "3", 4), my_func_safe)
```

```
## [1] "odd"  "even" NA      "even"
```



See a more realistic example when we talk about Web Scraping



What would `map_chr(c(1, 2, "3", 4), my_func_safe)` return?

# walk(), walk2(), pwalk()

You don't always need to *return* anything, you just wanna loop

```
walk(1:10, ~print("Hey Girrrl"))
```

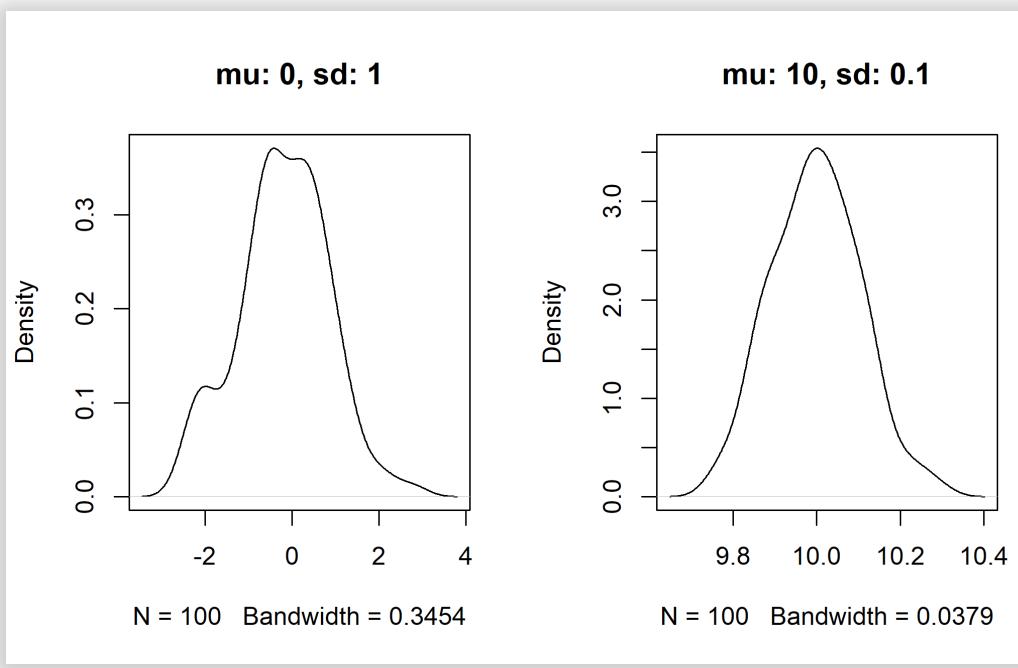
```
## [1] "Hey Girrrl"
```

```

plot_norm <- function(mu, sd) {
  plot(density(rnorm(100, mu, sd)), 
        main = str_c("mu: ", mu, ", sd: ", sd))
}

par(mfcol = c(1, 2))
walk2(c(0, 10), c(1, 0.1), plot_norm)

```



## In the words of Hadley

Once you master these functions, you'll find it takes much less time to solve iteration problems.

But you should never feel bad about using a for loop instead of a map function.

The important thing is that you solve the problem that you're working on, not write the most concise and elegant code.

Some people will tell you to avoid for loops because they are slow. They're wrong!