APPLICATIONS

OF DATA SCIENCE

Tidy Data Wrangling - Part B

Applications of Data Science - Class 3

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

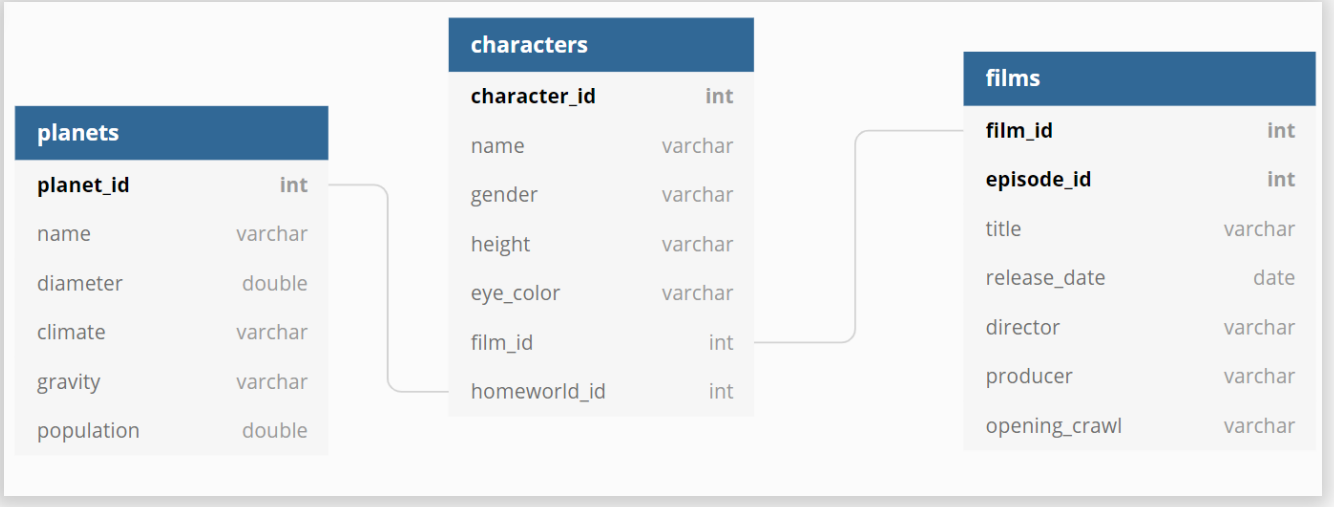
Stat. and OR Department, TAU

2022-12-25

Joining Tables

Detour: The Starwars Dataset(s)

The Starwars Dataset(s)




```
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, 2, 3, 3, 3, 3, 3, 3,
## $ name         <chr> "Luke Skywalker", "Luke Skywalker", "Luke Skywalker
## $ gender       <chr> "male", "male", "male", "male", "male", NA, NA, NA,
## $ homeworld_id <dbl> 1, 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, 1
## $ mass         <dbl> 77, 77, 77, 77, 77, 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)
```

```
## Rows: 61
## Columns: 10
## $ planet_id       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1
## $ name            <chr> "Tatooine", "Alderaan", "Yavin IV", "Hoth", "Dag
## $ rotation_period <dbl> 23, 24, 24, 23, 23, 12, 18, 26, 24, 27, 30, 27,
## $ orbital_period  <dbl> 304, 364, 4818, 549, 341, 5110, 402, 312, 368, 4
## $ diameter        <dbl> 10465, 12500, 10200, 7200, 8900, 118000, 4900, 1
## $ climate         <chr> "arid", "temperate", "temperate, tropical", "fro
## $ gravity         <chr> "1 standard", "1 standard", "1 standard", "1.1 s
## $ terrain         <chr> "desert", "grasslands, mountains", "jungle, rain
## $ surface_water   <dbl> 1.0, 40.0, 8.0, 100.0, 8.0, 0.0, 8.0, 12.0, NA,
## $ population      <dbl> 2.0e+05, 2.0e+09, 1.0e+03, NA, NA, 6.0e+06, 3.0e
```

```
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 Marquan
## $ producer      <chr> "Gary Kurtz, Rick McCallum", "Gary Kutz, Rick McCa
## $ release_date  <chr> "1977-05-25", "1980-05-17", "1983-05-25", "1999-05
```

End of Detour

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, 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 Luca
## 2            1 Luke Skywalker       2 The Empire Strikes Back  Irvin Kersh
## 3            1 Luke Skywalker       3 Return of the Jedi       Richard Mar
## 4            1 Luke Skywalker       6 Revenge of the Sith      George Luca
## 5            1 Luke Skywalker       7 The Force Awakens        J. J. Abram
## 6            2 C-3PO                1 A New Hope               George Luca
## 7            2 C-3PO                2 The Empire Strikes Back  Irvin Kersh
```

(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, 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           23 blah blah blah                
## 2       1        1         2           15 something something           
## 3       1        1         2           23 something something to you to 
## 4       1        1         3           15 <NA>                          
## 5       1        1         3           23 <NA>                          
## 6       1        1         4            8 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?

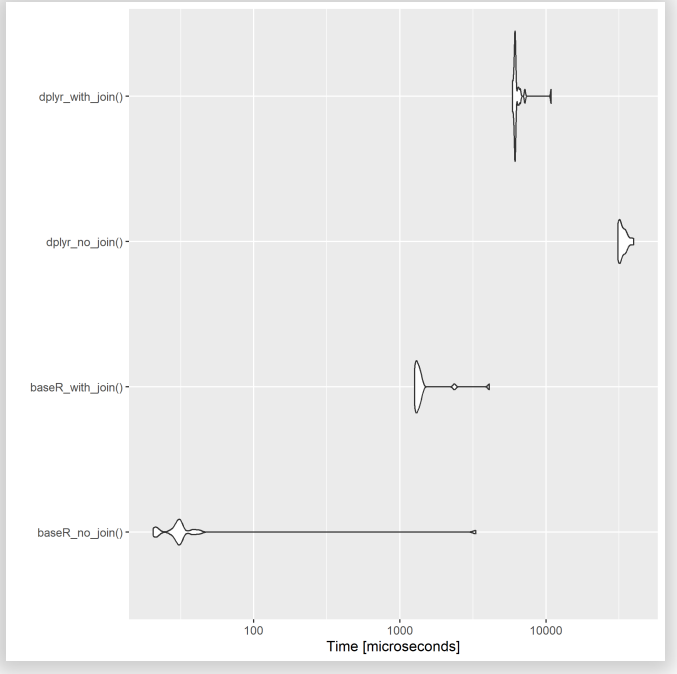
```
characters %>%
  inner_join(films %>% filter(director == "George Lucas"),
             by = "film_id") %>%
  inner_join(planets %>% filter(name == "Alderaan"),
             by = c("homeworld_id" = "planet_id"),
                    suffix = c("_char", "_planet")) %>%
  pull(name_char) %>%
  unique()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

```
## [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[fil
}

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

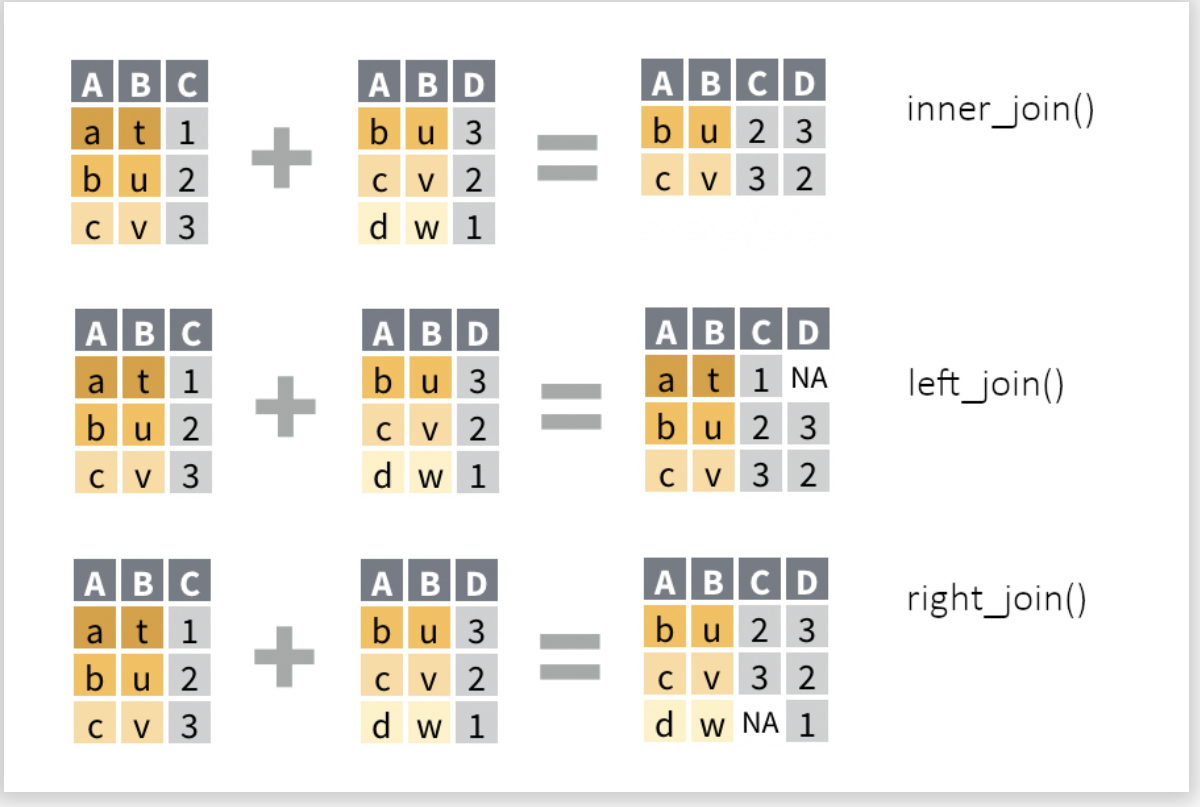
* Yes, I know, I answered my own question, 🤦

Other types of Joins?

Definitely, read here, here and here about:

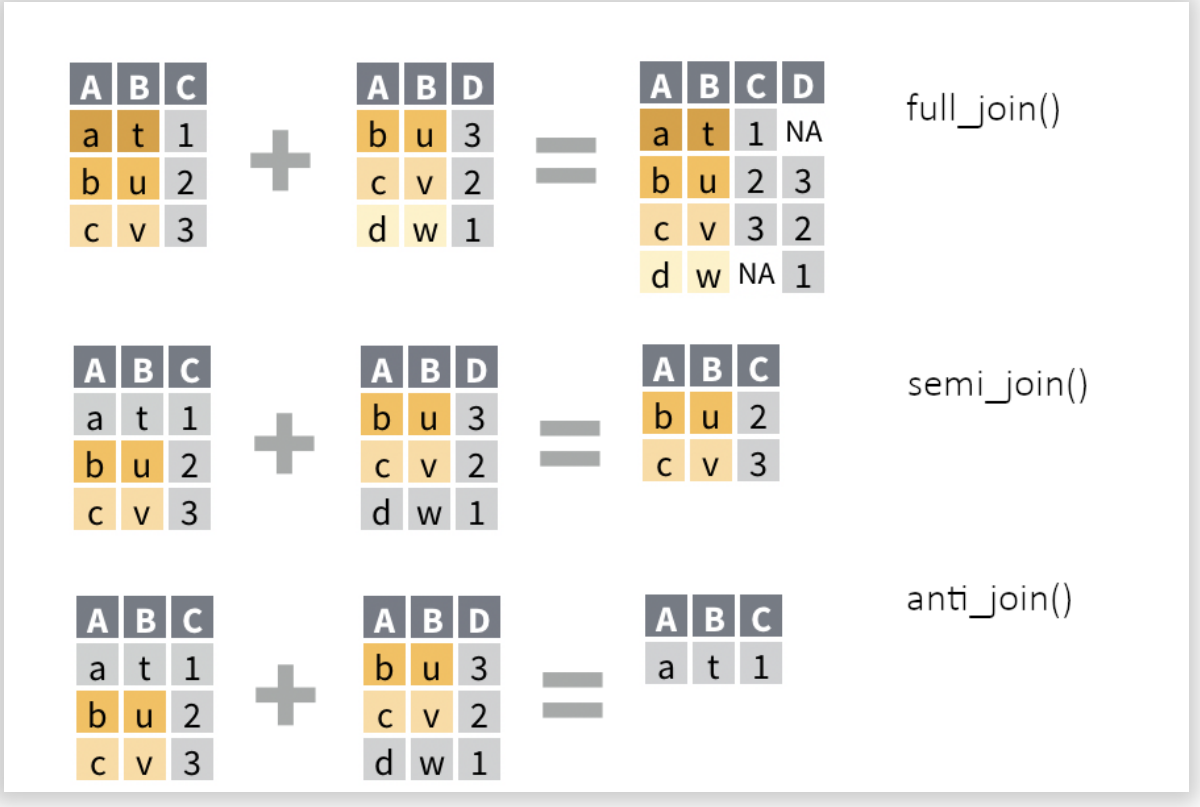
- `left_join()`
- `right_join()`
- `full_join()`
- `anti_join()`
- `semi_join()`

All Joins



Source: [The Tidyverse Cookbook]

All Joins



Source: [The Tidyverse Cookbook]

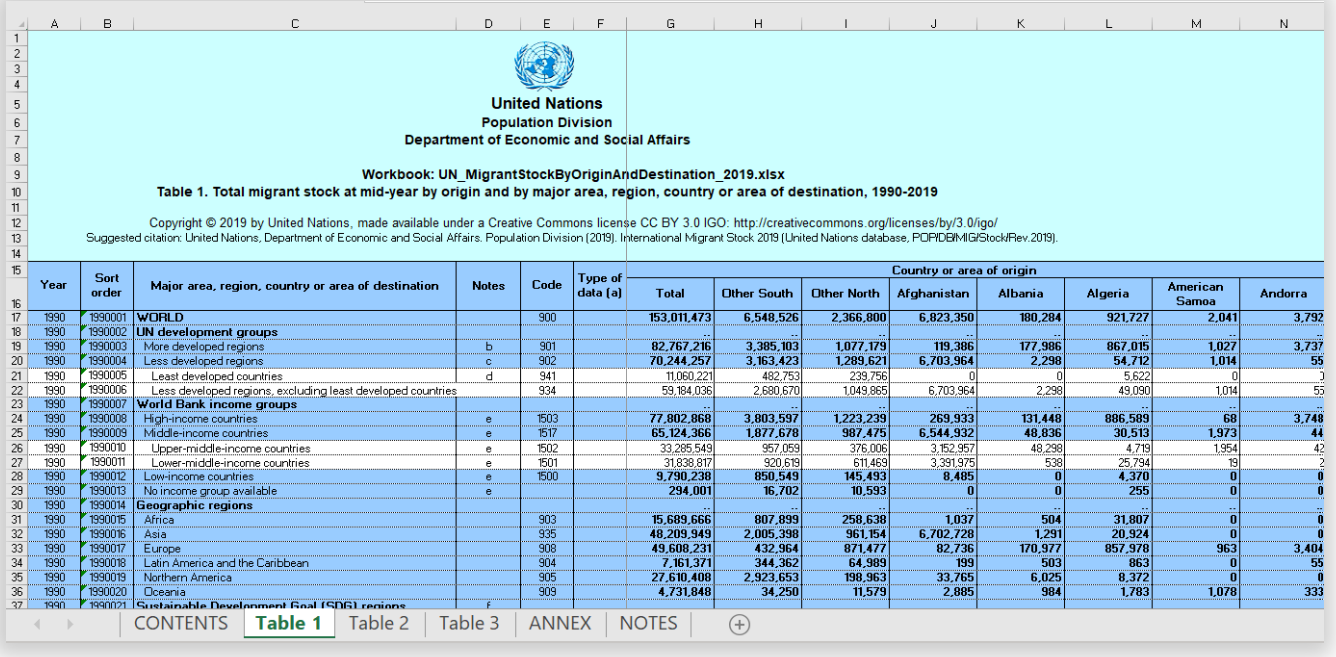
Tidying Tables

Detour: The Migration Dataset

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!



United Nations
Population Division
Department of Economic and Social Affairs

Workbook: UN_MigrantStockByOriginAndDestination_2019.xlsx
Table 1. Total migrant stock at mid-year by origin and by major area, region, country or area of destination, 1990-2019

Suggested citation: United Nations, Department of Economic and Social Affairs. Population Division (2019). International Migrant Stock 2019 (United Nations database, POP/DB/MIG/Stock/Rev.2019).

Year	Sort order	Major area, region, country or area of destination	Notes	Code	Type of data (a)	Country or area of origin							
						Total	Other South	Other North	Afghanistan	Albania	Algeria	American Samoa	Andorra
1990	1990001	**WORLD**		900		**153,011,473**	**6,548,526**	**2,366,800**	**6,823,350**	**180,284**	**921,727**	**2,041**	**3,792**
1990	1990002	**UN development groups**				..	..	..	..	..	..	..	..
1990	1990003	More developed regions	b	901		**82,767,216**	**3,385,103**	**1,077,179**	**119,386**	**177,986**	**867,015**	**1,027**	**3,737**
1990	1990004	Less developed regions	c	902		**70,244,257**	**3,163,423**	**1,289,621**	**6,703,964**	**2,298**	**54,712**	**1,014**	**55**
1990	1990005	Least developed countries	d	941		11,060,221	482,753	239,756	0	0	5,622	0	0
1990	1990006	Less developed regions, excluding least developed countries		934		59,184,036	2,680,670	1,049,865	6,703,964	2,298	49,090	1,014	55
1990	1990007	**World Bank income groups**				..	..	..	..	..	..	..	..
1990	1990008	High-income countries	e	1503		**77,802,868**	**3,803,597**	**1,223,239**	**269,933**	**131,448**	**886,589**	**68**	**3,748**
1990	1990009	Middle-income countries	e	1517		**65,124,366**	**1,877,678**	**987,475**	**6,544,932**	**48,836**	**30,513**	**1,973**	**44**
1990	1990010	Upper-middle-income countries	e	1502		33,285,549	957,059	376,006	3,152,957	48,298	4,719	1,954	42
1990	1990011	Lower-middle-income countries	e	1501		31,838,817	920,619	611,469	3,391,975	538	25,794	19	2
1990	1990012	Low-income countries	e	1500		**9,790,238**	**850,549**	**145,493**	**8,485**	**0**	**4,370**	**0**	**0**
1990	1990013	No income group available	e			**294,001**	**16,702**	**10,593**	**0**	**0**	**255**	**0**	**0**
1990	1990014	**Geographic regions**				..	..	..	..	..	..	..	..
1990	1990015	Africa		903		**15,689,666**	**807,899**	**258,638**	**1,037**	**504**	**31,807**	**0**	**0**
1990	1990016	Asia		935		**48,209,949**	**2,005,398**	**961,154**	**6,702,728**	**1,291**	**20,924**	**0**	**0**
1990	1990017	Europe		908		**49,608,231**	**432,964**	**871,477**	**82,736**	**170,977**	**857,978**	**963**	**3,404**
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		**27,610,408**	**2,923,653**	**198,963**	**33,765**	**6,025**	**8,372**	**0**	**0**
1990	1990020	Oceania		909		**4,731,848**	**34,250**	**11,579**	**2,885**	**984**	**1,783**	**1,078**	**333**
1990	1990021	**Sustainable Development Goal (SDG) regions**	f										

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-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

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>,
```

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, gende
  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")
```

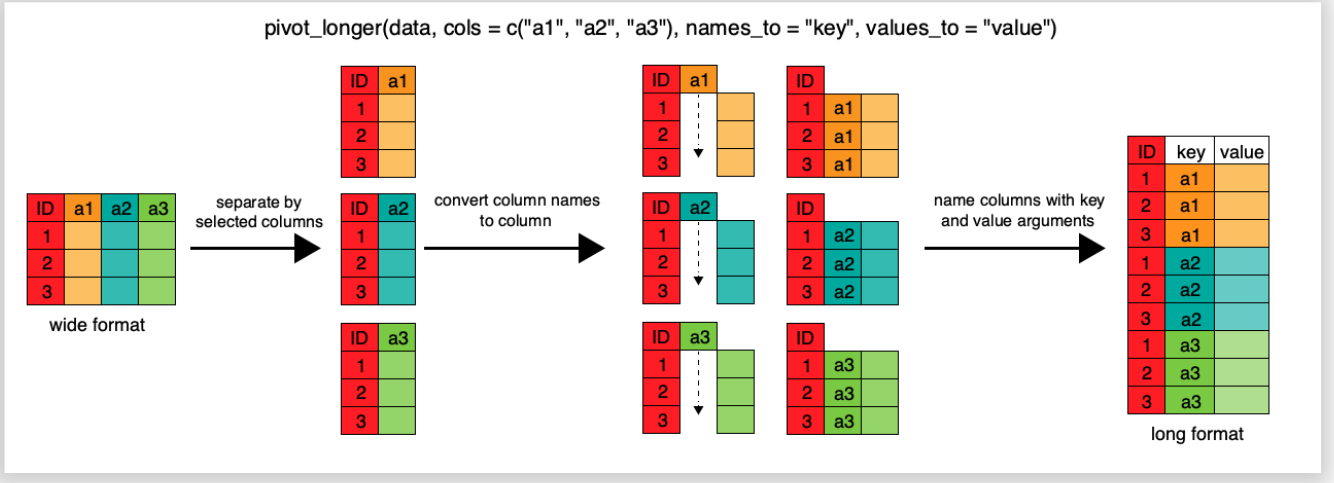
End of Detour

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]

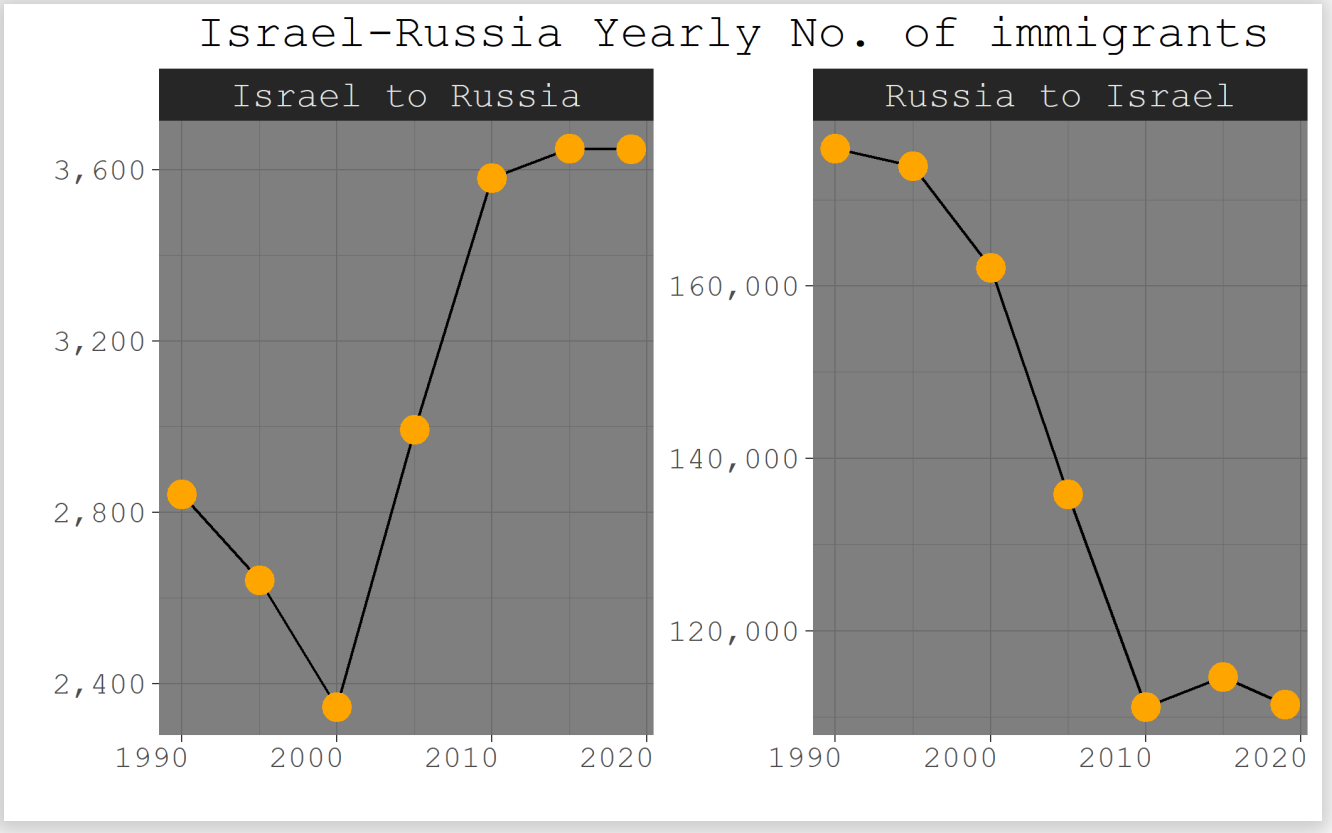
pivot_wider()

```
migration_wide <- migration_long %>%
  pivot_wider(id_cols = 1:4,
              names_from = country_orig,
              values_from = n_migrants)

migration_wide
```

```
## # 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>,
```

Where is this going to?

```
get_1way_migration <- function(.country_dest, .country_orig) {
  migration_long %>%
    filter(country_dest == .country_dest, country_orig == .country
    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 immigr
  theme_dark() +
  facet_wrap(~direction, scales = "free", labeller = labeller(dire
  scale_y_continuous(labels = scales::comma_format()) +
  theme(axis.text = element_text(size = 12, hjust=0.9, family = "n
        strip.text.x = element_text(size = 14, family = "mono"),
        plot.title = element_text(hjust = 0.5, size = 18, family =
```


Israel-Russia Yearly No. of immigrants
Israel to Russia
Russia to Israel
3,600
3,200
2,800
2,400
160,000
140,000
120,000
1990
2000
2010
2020

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()

```
table3_tidy <- table3 %>%
  separate(pct_straight, into = c("straight", "total"), sep = "/")

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 `purrr`)

```
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

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() 2701 2802 48088.16   2901 3101 4493702   100
##  nchar_vectorized()  800  901 13575.99    902 1001 1259601   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
`map()`	`map2()`	`pmap()`	`list`	`?`
`map_lgl()`	`map2_lgl()`	`pmap_lgl()`	`vector`	`logical`
`map_chr()`	`map2_chr()`	`pmap_chr()`	`vector`	`character`
`map_int()`	`map2_int()`	`pmap_int()`	`vector`	`integer`
`map_dbl()`	`map2_dbl()`	`pmap_dbl()`	`vector`	`double`
`map_dfr()`	`map2_dfr()`	`pmap_dfr()`	`tibble`	`?`

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_heigh

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          173. TRUE
## 2 f          165. TRUE
## 3 f          168. TRUE
## 4 m          183. TRUE
## 5 m          180. TRUE
## 6 m          183. TRUE
```

Example2: You could even supply args

```
is_above_average_height <- function(sex, height_cm, men_avg, women
  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_heigh
                            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          178. TRUE
## 2 f          175. TRUE
## 3 f          168. TRUE
## 4 m          170. FALSE
## 5 m          188. TRUE
## 6 m          178. FALSE
```

Example2: Anonymous Functions

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

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 undebuggable 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>  <chr>
## 1 "about me:<br … "curr… "maki… "the … "book… "food… duali… "tryi… "i am.
## 2 "i am a chef: … "dedi… "bein…  <NA>  "i am… "deli… <NA>    <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 <in
## #   essay6_len <int>, essay7_len <int>, essay8_len <int>, essay9_len <in
```

`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 `essay`s:

```
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 essay
##   <chr>                                                     <chr>  <chr>
## 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>        <l
##  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_heigh
```

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?

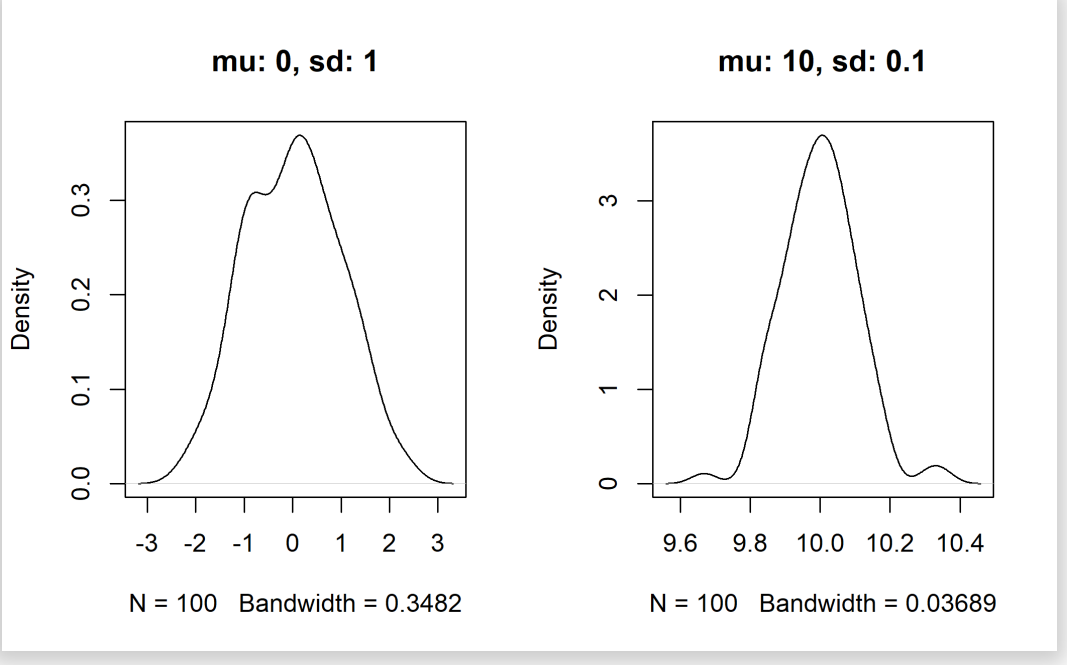
walk(),walk2(),pwalk()

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

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

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