MODUL 9
DATA WRAGGLING
(RESHAPING DATA, JOINING TABLE)

A. Tujuan Praktikum

- Menggunakan fungsi `gather` dan `spread` untuk *reshaping data*
- Menggunakan fungsi `left_join`, `right_join`, `inner_join`, `full_join`, `semi_join`, dan `anti_join` untuk menggabungkan data dari beberapa tabel

B. Alokasi Waktu

1 x pertemuan = 120 menit

C. Dasar Teori

Dalam suatu proyek *data science*, akan sangat jarang kita menemui data yang siap olah atau telah tersedia dalam satu paket/ satu kesatuan data. Oleh karena itu, kita akan selalu melakukan sedikit pekerjaan "di belakang layar" terlebih dahulu setelah mengimpor data yang akan kita olah ke *workspace*. *Preprocessing* dibutuhkan untuk menjadikan data bersifat *tidy*. Biasanya, data yang akan kita olah akan berada dalam file, *database*, atau diekstraksi dari dokumen, termasuk halaman *web*, *tweet*, atau PDF. Dalam modul ini, setelah proses impor data ke R selanjutnya, kita akan menggunakan `tidyverse` untuk merapikan data. Langkah awal dalam proses analisis data ini biasanya melibatkan beberapa, langkah-langkah untuk mentransformasi data dari bentuk mentah ke bentuk *tidy* yang sangat akan memudahkan proses analisis. Kita akan menyebut proses ini sebagai *data wraggling*.

Reshaping Data

Seperti yang telah kita bahas pada modul sebelumnya, memiliki data dalam format *tidy* adalah kondisi yang harus terpenuhi sebelum menggunakan `tidyverse`. Setelah langkah pertama dalam proses analisis data, yaitu: mengimpor data, langkah selanjutnya yang umum dilakukan adalah membentuk kembali data menjadi bentuk yang memfasilitasi proses analisis. Paket `tidyr` memiliki beberapa fungsi yang dapat kita gunakan untuk merapikan data. Pada bagian ini, kita akan menggunakan *dataset*: "*fertility-two-countries-example"* sebagai contoh kasus.

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

1. `gather`

Salah satu fungsi yang paling sering digunakan dalam paket `tidyr` adalah `gather`, yang berguna untuk mengubah *wide-data* menjadi data yang bersifat *tidy*.

Seperti kebanyakan fungsi `tidyverse`, argumen pertama `gather` membutuhkan *input* berupa *data frame* yang akan ditransformasi. Di sini kita akan melakukan *reshaping data* dari *dataset wide_data* sehingga setiap baris akan berisi nilai pengamatan fertilitas, dan terdapat tiga kolom untuk menyimpan tahun, negara, dan nilai yang diamati. Dalam *wide_data* saat ini, data dari tahun yang berbeda ditampilkan dalam bentuk kolom yang berbeda dengan tahun sebagai nama

kolom. Pada argumen kedua dan ketiga kita akan memberitahu `gather` nama kolom yang kita inginkan, yaitu: `year` dan `fertility`. Sebelumnya tidak ada kolom atau variabel dalam *wide_data* yang menyinggung bahwa data angka yang ditampilkan merupakan nilai fertilitas. Hal tersebut, secara eksplisit dapat kita ketahui dari nama file. Maka dari itu, pada argumen keempat kita akan membuat kolom yang berisi nilai fertilitas. Pada kolom inilah fungsi `gather` akan menyimpan hasil *reshaping data*. Umumnya kita juga perlu mendefiniskan kolom mana yang nilainya akan "dikumpulkan", dalam contoh data yang kita miliki, kita akan "mengumpulkan" kolom 1960 hingga 2015.

Script untuk *reshaping data* fertilitas yang digunakan adalah sebagai berikut:

```
new_tidy_data <- gather(wide_data, year, fertility, `1960`:`2015`)
```

Kita juga dapat menggunakan operator *pipe* seperti contoh ini:

```
new_tidy_data <- wide_data %>% gather(year, fertility, `1960`:`2015`)
```

Hasil transformasi data menjadi bentuk yang *tidy* akan menghasilkan kolom '*year*' dan '*fertility*':

```
head(new_tidy_data)
#> # A tibble: 6 x 3
#> country year fertility
#> <chr> <chr> <dbl>
#> 1 Germany 1960 2.41
#> 2 South Korea 1960 6.16
#> 3 Germany 1961 2.44
#> 4 South Korea 1961 5.99
#> 5 Germany 1962 2.47
#> # ... with 1 more row
```

Dari hasil transformasi diatas, setiap tahun akan menghasilkan dua baris data, karena kita hanya memiliki dua negara dan kolom negara tidak "dikumpulkan". Alternatif lain, *script* diatas juga akan menghasilkan output yang dengan *script* berikut:

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country)
```

Bentuk objek '*new_tidy_data*' terlihat seperti '*tidy_data*' yang kita definisikan dengan cara ini

```
data("gapminder")
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany") & !is.na(fertility)) %>%
  select(country, year, fertility)
```

Namun, terdapat satu perbedaan penting antara keduanya, yaitu: tipe data kolom '*year*':

```
class(tidy_data$year)
#> [1] "integer"
class(new_tidy_data$year)
#> [1] "character"
```

Fungsi `gather` mengasumsikan bahwa nama kolom memiliki tipe data karakter, sehingga tahun yang seharusnya bertipe numerik juga dianggap memiliki tipe sama dengan judul kolomnya. Maka dari itu, kita perlu melakukan *data wraggling* lagi sebelum kita membuat plot. Kita akan

mengkonversi kolom tahun menjadi menjadi tipe data numerik. Pada *script* selanjutnya, kita tambahkan argumen `convert` pada fungsi `gather`:

```
new_tidy_data <- wide_data %>%
  gather(year, fertility, -country, convert = TRUE)
class(new_tidy_data$year)
#> [1] "integer"
```

Opsi lain yang dapat digunakan untuk melakukan konversi adalah dengan menggunakan fungsi `mutate` dan `as.numeric`.

Sekarang kita telah memiliki data *tidy*, sehingga kita dapat menggunakan fungsi `ggplot` untuk memvisualisasikan '*new_tidy_data*':

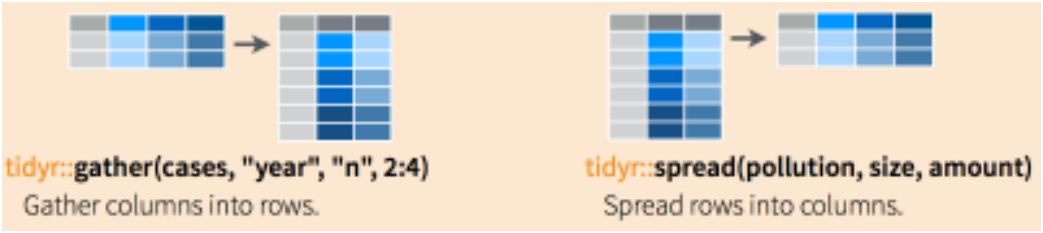
```
new_tidy_data %>% ggplot(aes(year, fertility, color = country)) +
  geom_point()
```

2. `spread`

Contoh kasus pada bagian ini merupakan kebalikan dari contoh kasus sebelumnya. Kita akan menggunakan fungsi `spread` untuk mengubah data *tidy* menjadi *wide_data*. Biasanya proses ini sering digunakan sebagai langkah perantara dalam merapikan data. Fungsi `spread` pada dasarnya adalah kebalikan dari fungsi `gather`. Argumen pertama berisi data yang akan dikonversi. Argumen kedua digunakan untuk memberitahu `spread` variabel mana yang akan menjadi nama kolom. Argumen ketiga menentukan variabel mana yang digunakan untuk mengisi sel:

```
new_wide_data <- new_tidy_data %>% spread(year, fertility)
select(new_wide_data, country, `1960`:`1967`)
#> # A tibble: 2 x 9
#> country `1960` `1961` `1962` `1963` `1964` `1965` `1966` `1967`
#> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 Germany 2.41 2.44 2.47 2.49 2.49 2.48 2.44 2.37
#> 2 South Korea 6.16 5.99 5.79 5.57 5.36 5.16 4.99 4.85
```

Sebagai ringkasan, gambar berikut merupakan ilustrasi dari cara kerja kedua fungsi yang telah kita diskusikan sebelumnya:



*Gambar milik RStudio

Joining Table

Informasi yang kita butuhkan untuk analisis data mungkin tidak hanya melibatkan satu tabel. Misalnya, untuk melakukan *forecasting* hasil pemilihan (*election*), kita dapat menggunakan fungsi `left_join` untuk menggabungkan informasi dari tabel populasi dan tabel hasil pemilihan di tahun sebelumnya. Untuk mempermudah mengenai pemahaman fungsi *joining table*, kita akan

menggunakan contoh yang lebih sederhana untuk menggambarkan langkah-langkah proses penggabungan tabel.

Misalkan, kita ingin mengeksplorasi hubungan antara jumlah populasi negara bagian AS terhadap pemilihan pemilih. Data populasi kita peroleh melaui tabel ini:

```
library(tidyverse)
library(dslabs)
data(murders)
head(murders)
#> state abb region population total
#> 1 Alabama AL South 4779736 135
#> 2 Alaska AK West 710231 19
#> 3 Arizona AZ West 6392017 232
#> 4 Arkansas AR South 2915918 93
#> 5 California CA West 37253956 1257
#> 6 Colorado CO West 5029196 65
```

dan suara pemilih dalam tabel berikut:

```
head(results_us_election_2016)
#> state electoral_votes clinton trump others
#> 1 California 55 61.7 31.6 6.7
#> 2 Texas 38 43.2 52.2 4.5
#> 3 Florida 29 47.8 49.0 3.2
#> 4 New York 29 59.0 36.5 4.5
#> 5 Illinois 20 55.8 38.8 5.4
#> 6 Pennsylvania 20 47.9 48.6 3.6
```

Penggabungan biasa kedua tabel tersebut tidak akan berhasil karena urutan negara tidak sama.

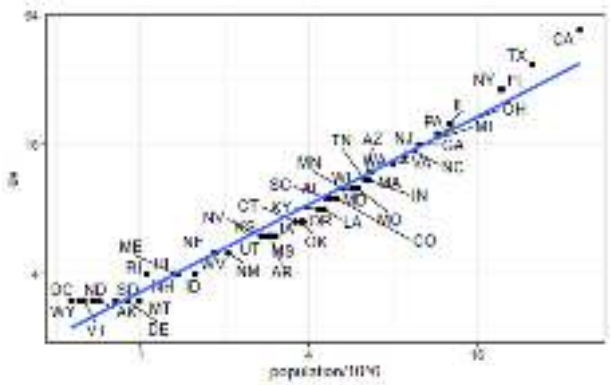
```
identical(results_us_election_2016$state, murders$state)
#> [1] FALSE
```

Fungsi yang dapat digunakan untu penggabungan tabel data dalam paket `dplyr` memastikan bahwa tiap baris dari tabel yang digabungkan nilainya cocok satu sama lain. Sama halnya dengan menggunakan SQL, pendekatan dan sintaks R yang digunakan untuk fungsi penggabungan sangat mirip. Prinsip utama yang digunakan adalah bahwa perlu diidentifikasi terlebih dahulu satu atau lebih kolom yang sesuai antar tabel. Kemudian tabel baru dengan informasi gabungan akan dibentuk. Kedua tabel diatas kemudian akan kita gabungkan menggunakan fungsi `left_join`. Perhatikan hasil penggabungan berikut.

```
tab <- left_join(murders, results_us_election_2016, by = "state") %>%
select(-others) %>% rename(ev = electoral_votes)
head(tab)
#>   state abb region population total ev clinton trump
#> 1 Alabama AL South 4779736 135 9 34.4 62.1
#> 2 Alaska AK West 710231 19 3 36.6 51.3
#> 3 Arizona AZ West 6392017 232 11 45.1 48.7
#> 4 Arkansas AR South 2915918 93 6 33.7 60.6
#> 5 California CA West 37253956 1257 55 61.7 31.6
#> 6 Colorado CO West 5029196 65 9 48.2 43.3
```

Setelah data berhasil digabungkan, kemudian untuk analisa lebih lanjut, misalnya, kita dapat membuat plot untuk mengeksplorasi pola data baru yang kita miliki:

```
library(ggrepel)
tab %>% ggplot(aes(population/10^6, ev, label = abb)) +
geom_point() +
geom_text_repel() +
scale_x_continuous(trans = "log2") +
scale_y_continuous(trans = "log2") +
geom_smooth(method = "lm", se = FALSE)
```



Dalam praktiknya, tidak selalu tiap baris dalam satu tabel memiliki baris yang cocok dengan tabel yang lain. Maka dari itu, terdapat beberapa opsi fungsi *join* yang dapat digunakan. Untuk memudahkan identifikasi perbedaan hasil dari tiap fungsi yang akan kitab bahas satu persatu, kita akan mengambil himpunan bagian dari tabel yang kita gunakan di atas. Pada *script* berikutnya, kita akan membuat objek '*tab1*' dan '*tab2*' yang isinya terdiri dari beberapa *state* yang sama, tetapi tidak sama secara keseluruhan:

```
tab_1 <- slice(murders, 1:6) %>% select(state, population)
tab_1
#>   state      population
#> 1 Alabama    4779736
#> 2 Alaska     710231
#> 3 Arizona    6392017
#> 4 Arkansas   2915918
#> 5 California 37253956
#> 6 Colorado   5029196

tab_2 <- results_us_election_2016 %>%
filter(state%in%c("Alabama", "Alaska", "Arizona",
"California", "Connecticut", "Delaware")) %>%
select(state, electoral_votes) %>% rename(ev = electoral_votes)
tab_2
#>   state       ev
#> 1 California  55
#> 2 Arizona     11
#> 3 Alabama     9
#> 4 Connecticut 7
#> 5 Alaska      3
#> 6 Delaware    3
```

Kita akan menggunakan dua tabel ini sebagai contoh data di bagian selanjutnya.

1. `left_join`

Misalkan kita menginginkan tabel yang isinya mirip dengan '*tab_1*', dan menambahkan kolom *electoral_votes* (ev) pada tabel baru kita. Disini kita akan menggunakan `left_join` terhadap '*tab_1*' sebagai argumen pertama. Kita juga akan menambahkan argumen untuk menentukan kolom mana yang akan digunakan sebagai dasar penggabungan. Dalam contoh ini, kita akan menggunakan kolom *state* sebagai argumen dasar penggabungan.

```
left_join(tab_1, tab_2, by = "state")
#>   state      population ev
#> 1 Alabama    4779736    9
#> 2 Alaska     710231     3
#> 3 Arizona    6392017    11
#> 4 Arkansas   2915918    NA
#> 5 California 37253956   55
#> 6 Colorado   5029196    NA
```

Berdasarkan hasil diatas, pada *state*: Arkansas dan Colorado, akan muncul nilai NA karena ke dua nama *state* tidak muncul di '*tab_2*'. Sebagai alternatif, fungsi penggabungan juga dapat dilakukan dengan operator *pipe*:

```
tab 1 %>% left join(tab 2, by = "state")
```

2. `right_join`

Contoh selanjutnya ini merupakan kebalikan dari contoh pertama. Disini kita ingin menggabungkan '*tab_1*' terhadap '*tab_2*', sehingga fungsi yang digunakan adalah `right_join`:

```
tab_1 %>% right_join(tab_2, by = "state")
#>   state       population ev
#> 1 California  37253956   55
#> 2 Arizona     6392017    11
#> 3 Alabama     4779736    9
#> 4 Connecticut NA         7
#> 5 Alaska      710231     3
#> 6 Delaware    NA         3
```

Sekarang NA akan muncul di kolom yang berasal dari '*tab_1*'.

3. `inner_join`

Jika kita hanya ingin menyimpan baris yang sama-sama berisi informasi yang sama di kedua tabel yang kita miliki, fungsi `inner_join` dapat digunakan. Logika yang digunakan kurang lebih sama dengan konsep *intersect* pada himpunan data:

```
inner_join(tab_1, tab_2, by = "state")
#>   state      population ev
#> 1 Alabama    4779736    9
#> 2 Alaska     710231     3
#> 3 Arizona    6392017    11
#> 4 California 37253956   55
```

4. `full_join`

Jika kita ingin menggabungkan keseluruhan baris pada kedua tabel dan mengisi bagian yang hilang dengan NA, kita dapat menggunakan `full_join`. Logika yang digunakan sama dengan konsep *union* pada himpunan data:

```
full_join(tab_1, tab_2, by = "state")
#>    state       population ev
#> 1 Alabama     4779736    9
#> 2 Alaska      710231     3
#> 3 Arizona     6392017    11
#> 4 Arkansas    2915918    NA
#> 5 California  37253956   55
#> 6 Colorado    5029196    NA
#> 7 Connecticut NA         7
#> 8 Delaware    NA         3
```

5. semi_join

Fungsi `semi_join` dapat digunakan untuk menampilkan bagian dari tabel pertama yang juga memiliki baris informasi yang sama di tabel kedua. Dengan `semi_join`, kita tidak akan menambahkan data apapun dari kolom kedua:

```
semi_join(tab_1, tab_2, by = "state")
#> state population
#> 1 Alabama 4779736
#> 2 Alaska 710231
#> 3 Arizona 6392017
#> 4 California 37253956
```

6. anti_join

Fungsi `anti_join` adalah kebalikan dari `semi_join`. Hasil dari fungsi ini akan menampilkan elemen dari tabel pertama yang tidak memiliki baris informasi yang sama di tabel kedua:

```
anti_join(tab_1, tab_2, by = "state")
#> state population
#> 1 Arkansas 2915918
#> 2 Colorado 5029196
```

Perbedaan antara enam fungsi join yang kita bahas diatas dapat dirangkum sesuai ilustrasi pada gambar berikut:

*Gambar milik RStudio

D. Latihan

Reshaping data

1. Jalankan perintah berikut untuk membuat objek baru bernama "*co2_wide*":

```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
setNames(1:12) %>%
mutate(year = as.character(1959:1997))
```

   Gunakan fungsi `gather` untuk mentransformasi *dataset* menjadi data *tidy*. Tampilkan hasil data tidy yang berhasil dibuat.

2. Plot CO2 versus *month* dengan plot yang berbeda untuk setiap *year* menggunakan *script* ini:

```
co2_tidy %>% ggplot(aes(month, co2, color = year)) + geom_line()
```

   Jika plot yang diharapkan tidak berhasil dibuat, kemungkinan penyebabnya adalah karena co2_tidy$month tipe datanya bukan numerik:

```
class(co2_tidy$month)
```

   Modifikasi ulang *script* `gather` sehingga *plot* yang diinginkan berhasil ditampilkan

Joining Table

Instal dan muat *library* Lahman. *Package* yang telah kita muat berisi *database* yang terkait dengan tim *baseball*. Di dalamnya, terdapat ringkasan statistik tentang bagaimana para pemain melakukan pelanggaran dan pertahanan selama beberapa tahun. Selain itu, terdapat pula informasi pribadi tentang para pemain.

1. *Data frame Batting* berisi statistik ofensif semua pemain selama beberapa tahun. Lakukan *preview* data, misalnya, tampilkan 10 *batting* teratas dengan menjalankan *script* ini:

```
library(Lahman)
top <- Batting %>%
filter(yearID == 2016) %>%
arrange(desc(HR)) %>%
slice(1:10)
top %>% as_tibble()
```

   Script diatas hanya menampilkan ID, bukan nama pemain. Nama-nama pemain dapat dilihat pada tabel ini

```
Master %>% as_tibble()
```

   Pada tabel tersebut, bisa dilihat bahwa nama pemain dapat diidentifikasi pada kolom *nameFirst* dan *nameLast*. Gunakan fungsi `left_join` untuk membuat tabel baru yang berisi ID pemain, nama depan, nama belakang, dan jumlah *home run* (SDM). Simpan hasil `left_join` pada objek baru

2. Dengan menggunakan *dataset* yang sama, gunakan *data frame* "*Salaries*" untuk menambahkan informasi gaji masing-masing pemain ke tabel yang telah dibuat dalam latihan no 1. Perhatikan bahwa besar gaji berbeda setiap tahunnya. Pastikan untuk memfilter gaji pada tahun 2016, lalu gunakan `right_join`. Tampilkan hasil yang berisi: nama depan, nama belakang, tim, SDM, dan gaji.